



NACAD

Núcleo Avançado de Computação de Alto Desempenho



AMT
High Performance Cloud Computing



Modernização de Códigos

Dr. José Camata
camata@nacad.ufrj.br

NACAD/COPPE-UFRJ
Núcleo Atendimento em Computação de Alto Desempenho
(Parceiro AMT – Centro de Inovação da Intel)

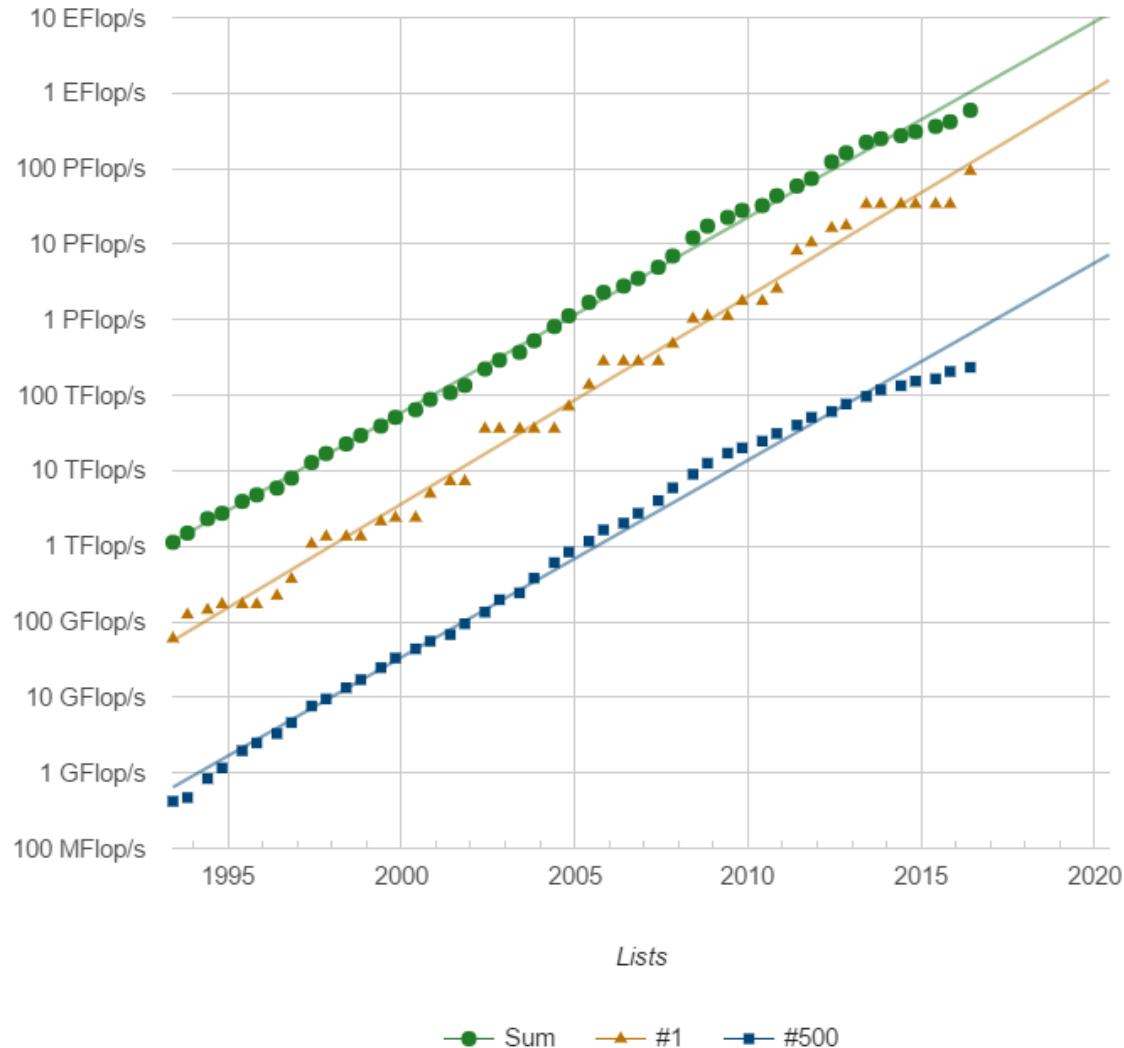


Agenda

- Modernização de Códigos
 - Arquitetura Intel
 - Processo de otimização
 - Métricas de desempenho e perfilagem
 - Técnicas de otimização:
 - manual e automática
 - Otimização Vetorial
 - Identificando Paralelismo Multi-Threading
 - Hands-on.

Computação na Era Exascale

Projected Performance Development





Desafios da Computação Exascale

- A arquitetura do processador em aberto.
- Consumo energético
 - Projeção simplista exigiria 200 MW em 2020, o que é insustentável. A meta é de 20-40 MW para 1 exaflop.
- Largura de banda de memória e capacidade não estão mantendo o mesmo ritmo do crescimento dos FLOPs.
- Diminuição da frequências do *clock* e o número de unidades de processamento num único chip devem aumentar.
 - Modelo de programação será revisto: compiladores não serão capazes de esconder o nível de concorrência das aplicações
- Não é esperado melhora significativa no custo de movimentação de dados tanto quanto a de flops.
 - Algoritmos necessitam minimizar o movimento de dados, e não flops.
- Aumento de complexidade do gerenciamento do sistema de I/O
- Confiabilidade e resiliência serão pontos fundamentais.

Arquitetura Intel

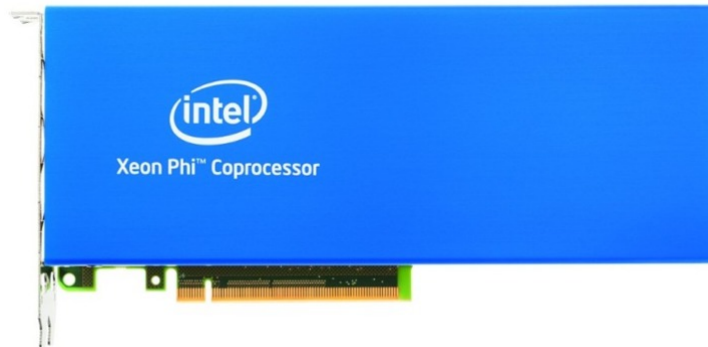
Intel Xeon
Processor



Current: Broadwell
Upcoming: Skylake

Multi-Core Architecture

Intel Xeon Phi
Coprorocessor, 1st generation



Knights Corner (KNC)

Intel Xeon Phi
Processor, 2nd generation*



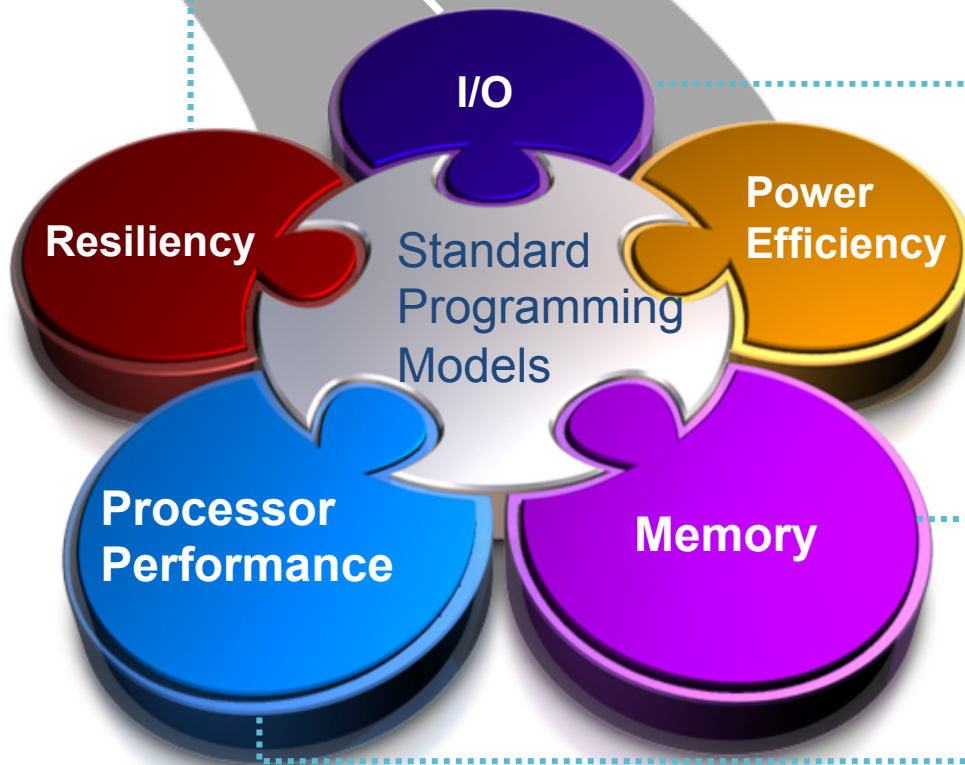
* socket and coprocessor versions

Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture

Next Step on Intel's Path to Exascale Computing

Exascale Vision



Next Step: KNL

*Systems scalable to
>100 PFlop/s*

*Up to 100 Gb/s with
Storm Lake integrated fabric*

Over 15 GF/Watt¹

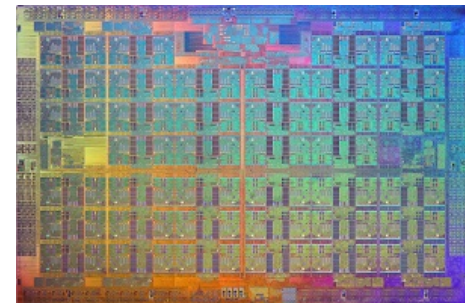
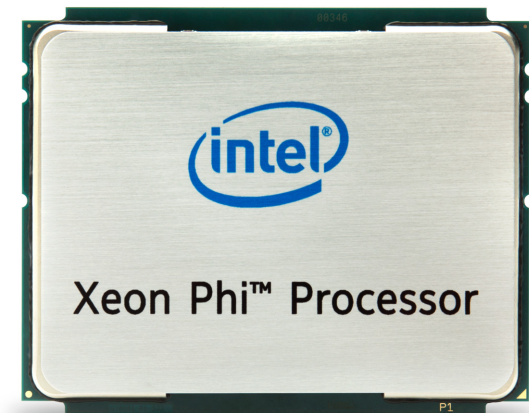
*~500 GB/s sustained memory
bandwidth with integrated on-
package memory*

*~3X Flops and ~3X single-
thread theoretical peak
performance over Knights
Corner¹*

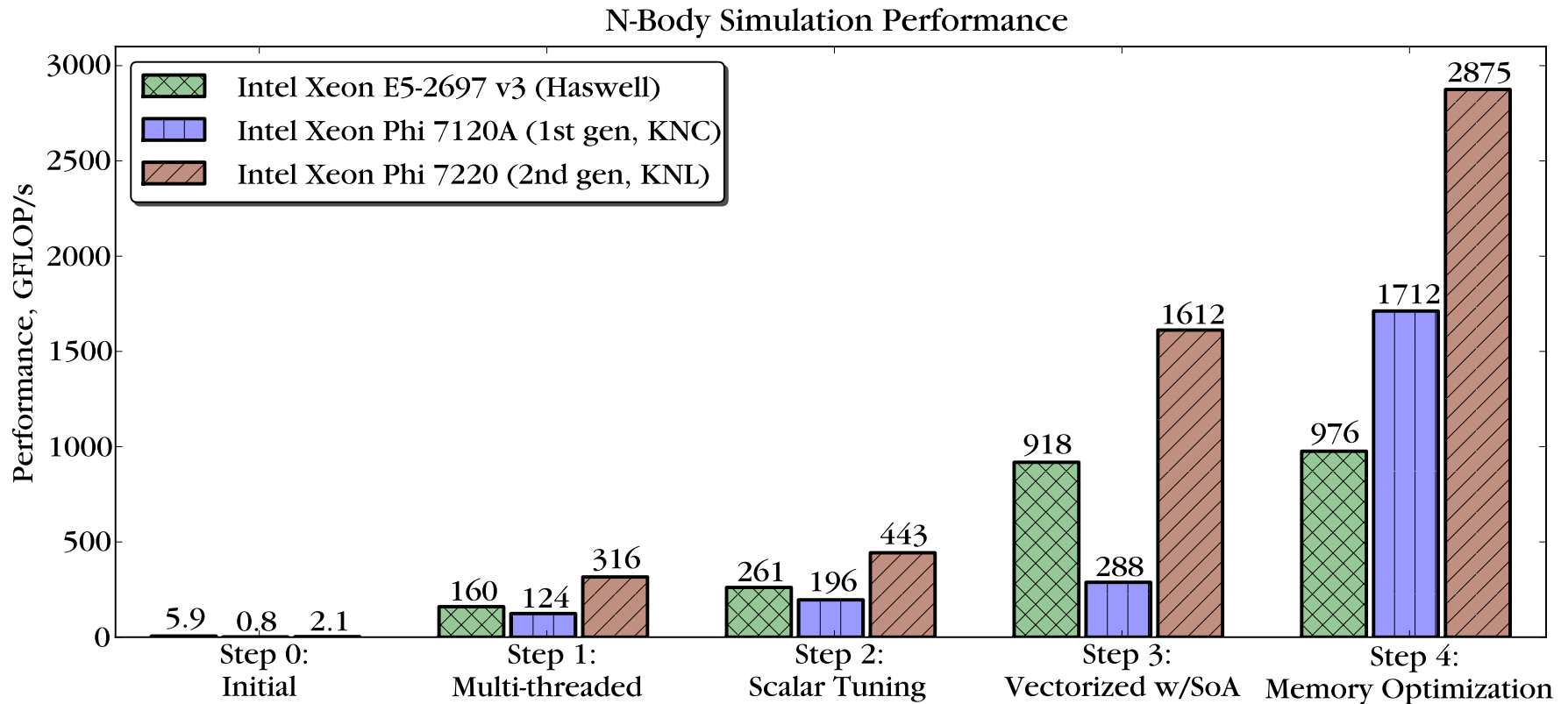


Intel Xeon Phi (2nd Gen)

- Plataforma especializada para aplicações que demandam alta computação
 - Versão em *socket* ou como coprocessador
 - 64-72 cores x 4 HT em 1.3-1.5GHz
 - +3 TFLOP/s in PD (FMA)
 - +6 TFLOP/s in PS (FMA)
 - > 380 GiB DDR4 (> 90 GB/s)
 - 16GiB HBM(MCDRAM, > 400 GB/s)
 - Binário compatível com Xeon



Por que modernização de código?



Detalhes sobre a simulação pode ser vista em: lotsofcores.com/KNLbook



NACAD

Núcleo Avançado de Computação de Alto Desempenho

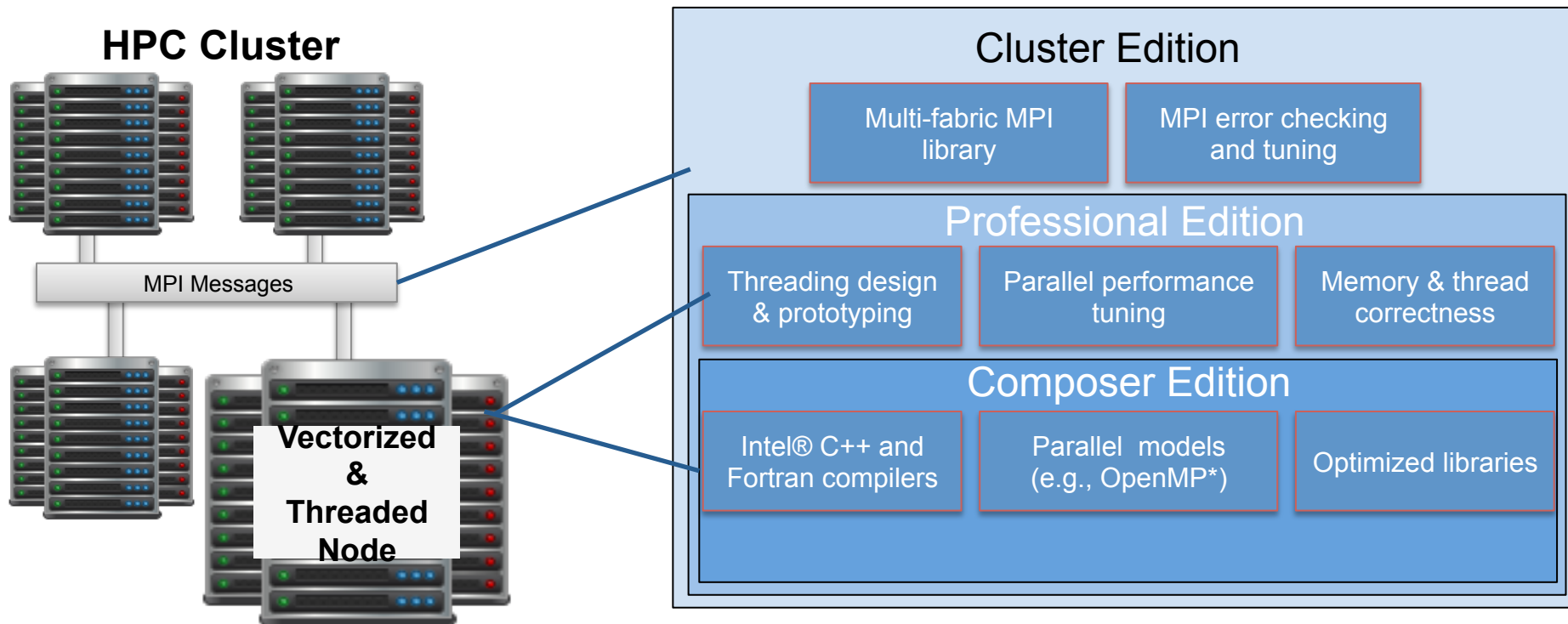


IDENTIFICANDO OPORTUNIDADES DE OTIMIZAÇÃO

Identificando oportunidades de otimização

Foco deste seminário: Modernização de código

Otimização em um único “nó” de processamento





Áreas de Otimização

Otimização Escalar

Pipelining está sendo usado?

Vetorização

SIMD está sendo bem usado?

Threading

Os cores cooperam eficientemente?

Memória

Uso de cache está maximizado?

Comunicação

A coordenação de mensagens em um sistema distribuído pode ser melhorada?



Ciclo de Otimização





Obter Informações de Desempenho

- **Perfilador:** programa capaz de reunir informações sobre o desempenho através de instrumentação de funções, métodos, blocos básicos e declarações.
 - Intel Vtune
 - PAPI
 - HPCToolkit, etc,...
- **Principais métricas de desempenho:**
 - CPU Time
 - Operações de Ponto-Flutuante por segundo FLOPS
 - Contadores de Hardwares: CPI, L1/L2 Cache Misses, etc

Intel® VTune™ Amplifier

- Get the data you need
 - Hotspot (Statistical call tree), call counts (statistical)
 - Thread profiling – concurrency and locks and waits analysis
 - Cache miss, bandwidth analysis...¹
 - GPU offload and OpenCL* kernel tracing
- Find answers fast
 - View results on the source/assembly
 - OpenMP* scalability analysis, graphical frame analysis
 - Filter out extraneous data – organize data with viewpoints
 - Visualize thread and task activity on the timeline
- Easy to use
 - No special compiles – C, C++, C#, Fortran*, Java*, ASM*
 - Visual Studio* integration or stand-alone
 - Graphical interface and command line
 - Local and remote data collection
 - Analyze Windows* and Linux* data on macOS*²

¹ Events vary by processor. ² No data collection on macOS*

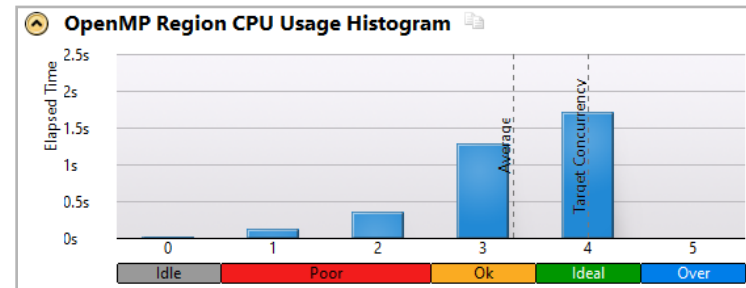
Quickly Find Tuning Opportunities

Function / Call Stack	CPU Time				Spin Time	Overhead Time
	Effective Time by Utilization	Idle	Poor	Ok		
FireObject::checkCollision	4.507s				0s	0s
FireObject::ProcessFireCollisionsRange	3.444s				0s	0s
NtWaitForSingleObject	0s				3.406s	0s
std::basic_ifstream<char,struct std::char_traits	3.359s				0s	0s
Ogre::FileSystemArchive::open	3.359s				0s	0s
CBaseDevice::Present	2.335s				0.671s	0s
Selected 1 row(s):					1.151s	0.728s

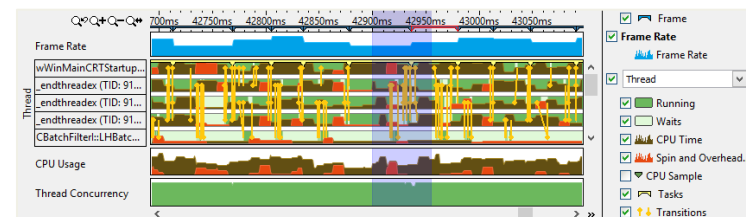
See Results On The Source Code

Source Line	Source	CPU Time: Total by Utilization
81	for (int i = 0; i < mem_array_i_max; i++)	0.300s
82	{	
83	for (int j = 0; j < mem_array_j_max; j++)	4.936s
84	{	
85	mem_array [j*mem_array_j_max+i] = *fill_val	7.207s

Tune OpenMP Scalability



Visualize & Filter Data





Técnicas de otimização

- Regras gerais:
 - Seleção dos melhores algoritmos;
 - Uso de bibliotecas eficientes (não esquecendo da portabilidade);
 - Exemplo: Álgebra Linear (Intel MKL)
 - Escolha do melhor layout de dados;
 - Programar com boas práticas de programação (otimizações manuais);
 - Usar otimizações do compilador **SEMPRE!!!**;



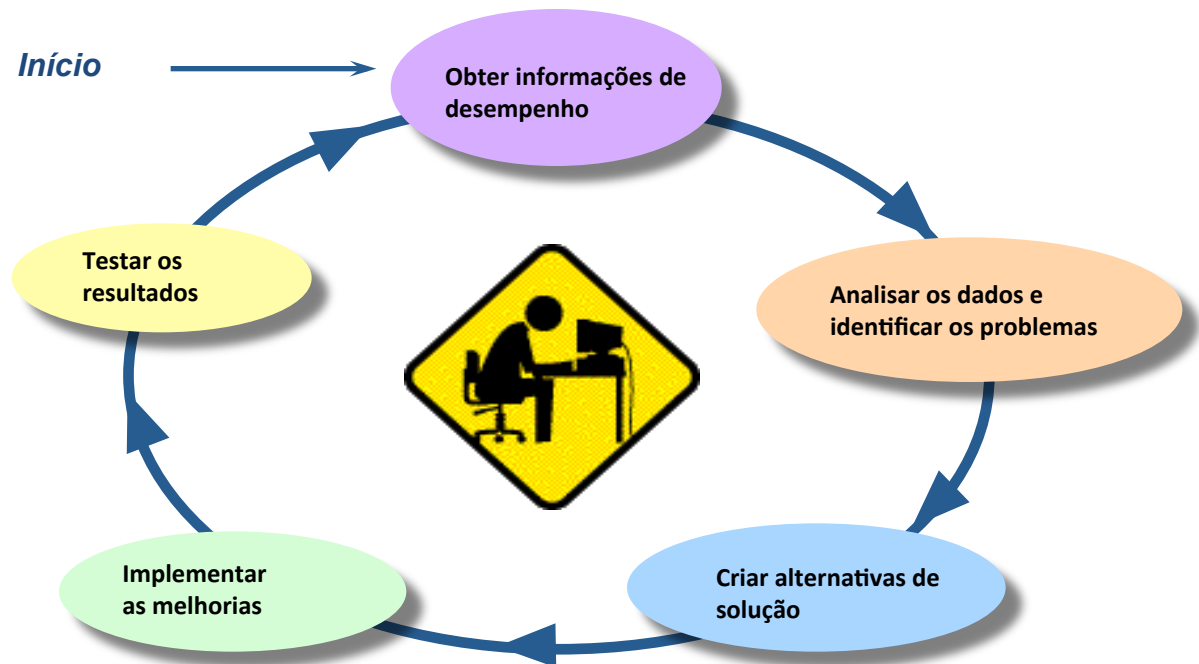
Técnicas de Otimização (cont)

- Otimização manual:
 - O usuário tem maior controle a respeito das dependências de dados;
 - Pode reduzir a legibilidade do programa;
 - Algumas otimizações podem ser benéficas em algumas arquiteturas e malélicas para outras;
- Otimização automática:
 - O compilador possui um conhecimento limitado das dependências de dados;
 - Praticamente impossível em C/C++ se forem utilizados ponteiros indiscriminadamente;
 - Não altera a legibilidade do código;
 - Resultados ótimos de acordo com a arquitetura e a habilidade do compilador;
 - Pode aumentar consideravelmente o tempo de compilação
- Otimização na prática
 - Realizar otimizações manuais básicas e deixar outras otimização por conta do compilador
 - Utilizar otimizações avançadas somente para as arquiteturas predominantes



Otimização Manual

- Forma mais trabalhosa porém a mais efetiva;
 - O programador deve ter controle sobre o que deve ser otimizado
- Baseia-se em:
 - Utilizar boas práticas de programação durante a fase de concepção do software;
 - Por em prática o ciclo de desempenho:





TÉCNICAS BÁSICAS DE OTIMIZAÇÃO



Reduções

Original

$2 * i$

onde i é um valor inteiro

Melhorado

$i + i$

Original

$2 * f$

onde f é um valor de ponto flutuante

Alternativa

$f + f$

essa modificação não traz benefícios pois a adição e a multiplicação de ponto flutuante consomem o mesmo número de ciclos

Original

$x ** 2 . d0$

Melhorado

No mínimo, substitua por:

$x ** 2$

ou melhor:

$x * x$



Eliminação de sub-expressões

Original

```
s1 = a + c + b  
s2 = a + b - c
```

ordem das operações inibindo
as otimizações por parte do
compilador

Original

```
r = 2.*x(i)  
s = x(i)-1.
```

Original

```
x = acos(-1.) + acos(-1.)**2
```

Melhorado

No mínimo, substitua por:

```
s1 = (a + b) + c  
s2 = (a + b) - c
```

ou melhor:

```
t = a + b  
s1 = t + c  
s2 = t - c
```

Melhorado

```
t1 = x(i)  
r = 2.*t1  
s = t1-1.
```

Melhorado

```
tt = acos(-1.)  
x = tt + tt**2
```

ou melhor:

```
tt = acos(-1.)  
x = tt + tt*tt
```

Muitas dessas otimizações podem ser automaticamente feitas pelo compilador



Expressões constantes em laços

Original

```
do i=1,n  
  a(i) = r*a(i)*s  
enddo
```

ordem das operações
inibindo as otimizações por
parte do compilador

Melhorado

No mínimo, substitua por:

```
do i=1,n  
  a(i) = r*s*a(i)  
enddo
```

ou melhor:

```
t1 = r*s  
do i=1,n  
  a(i) = t1*a(i)  
enddo
```



Conversão de sinais

- A conversão de sinais consome alguns ciclos de processamento;
- A omissão das conversões, em operações de ponto flutuante, podem alterar o resultado das operações, ou seja, o compilador não tenta fazer tais transformações em níveis baixos de otimização;

Original

```
a(i) = -(1.d0-.5d0*b(i))
```

Melhorado

```
a(i) = -1.d0+.5d0*b(i)
```



Propagação e avaliação de constantes e simplificação no cálculo de endereços

Original

```
two = 2.d0  
x = 3.d0*two*y
```

Melhorado

```
x = 6.d0*y
```

Original

```
dimension a(4,100)  
do i=1,100  
    x = x+a(1,i)  
enddo
```

o endereço de `a(1,i)` em relação ao elemento `a(1,1)` é `4*i-4`

Melhorado

```
dimension a(4*100)  
iadd = 1  
do i=1,100  
    x = x+a(iadd)  
    iadd = iadd+4  
enddo
```

aqui o endereço é calculado por uma simples adição



In-lining

- Função sendo chamada dentro de um laço:

```
...  
do i=1,max  
    r(i) = dist(x(i),y(i))  
enddo  
...  
function dist(x,y)  
real*8 :: x, y, dist  
dist = dsqrt(x*x + y*y)  
end function
```

- A técnica de “*in-lining*” consiste em substituir as chamadas de rotinas escrevendo-as explicitamente onde elas são chamadas:

```
...  
do i=1,max  
    r(i) = dsqrt(x(i)*x(i)+y(i)*y(i))  
enddo  
...
```

Os compiladores possuem flags específicos para realizar esse tipo de substituição baseado em alguns critérios, porém, quando efetuamos essa substituição explicitamente facilitamos o trabalho do compilador.



Precisão: Constantes

```
// bad: 2 is "int"
```

```
long b = a*2;
```

```
// bad: overflow
```

```
long n = 100000*100000;
```

```
// bad: excessive
```

```
float p=6.283185307179586;
```

```
// Bad: 2 is "int"
```

```
float q = 2*p;
```

```
// Bad: 1e9 is "double"
```

```
float r=1e9*p
```

```
// bad: 1 is "int"
```

```
double t=s+1
```

```
// Good: 2 is "long"
```

```
long b = a*2L;
```

```
// Good: correct
```

```
long n = 100000L*100000L;
```

```
// Good: accurate
```

```
float p=6.283185f;
```

```
// Good: 2 is "float"
```

```
float q = 2.0f*p;
```

```
// Goof: 1e9 is "float"
```

```
float r=1e9f*p
```

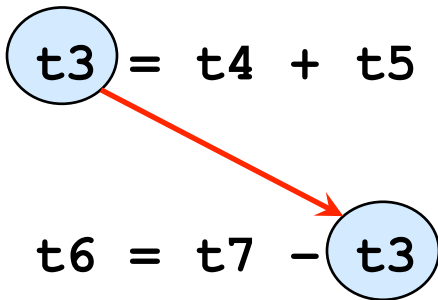
```
// Good: 1.0 is "double"
```

```
double t=s+1.0
```

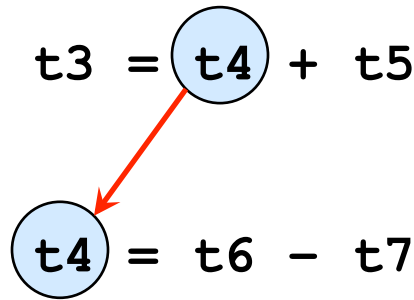


Tipos de dependências e instruções FMA

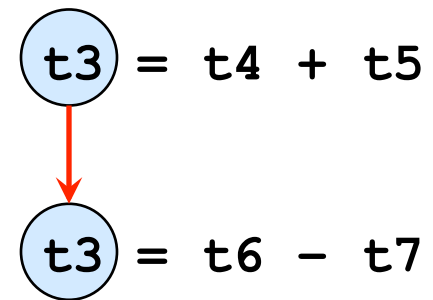
Dependência verdadeira



Anti-dependência



Dependência de saída



Fused multiply-add (FMA)

Original

```
x = 2.*y+1.
s = s+x
z = 3.*r+2.
t = t+z
```

Melhorado

```
x = 2.*y+1.
z = 3.*r+2.
s = s+x
t = t+z
```



Evite sempre!

- Testes no interior de laços intensivos

```
do i=1,1000
  if (i.lt.50) ! do something
enddo
```

- Chamadas de rotinas no interior de laços intensivos

```
do i=1,1000
  call something
enddo
```

- Operações de divisão “*...custam muito mais ciclos de processamento do que as operações de multiplicação...*”

- Operações de IO no interior de laços intensivos
- Variáveis não iniciadas;

- *Arranjos* com dimensões superiores a 2;

- Aritmética de arranjos do Fortran90 “*...facilita a vida do programador mas deixa o desempenho inteiramente sob responsabilidade do compilador.*”

- Sobrecarga de operadores (C++ e Fortran90)

- “Aliasing”: 2 arranjos compartilhando a mesma área de memória

```
– call sub(n, a(1), a(1), c(1), sum)
```



Otimização automática

- É a forma mais simples e rápida de se obter ganhos de desempenho, portanto, **UTILIZE SEMPRE** independentemente de ter feito otimização manual ou não;
- Para gerar um código otimizado o compilador realiza algumas operações bastante similares a algumas técnicas de otimização manual, tais como: alteração da ordem de algumas instruções, desenrolamento de laços, *inlining*, etc...



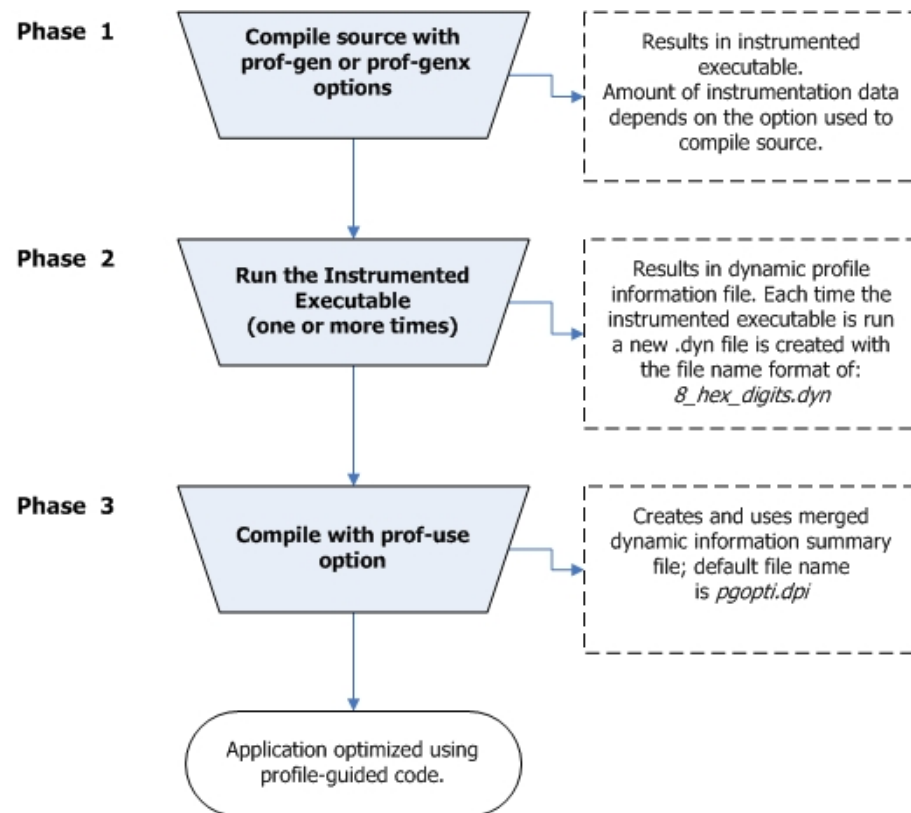
Otimização automática (cont.)

- Armadilhas dos flags de otimização:
 - Um compilador normalmente possui dezenas de opções de otimização que podem ser usadas individualmente ou agrupadas entre si. Nem sempre a associação dos flags de otimização produzem o melhor resultado. É importante ressaltar que em alguns casos uma associação de flags mal-feita pode acarretar em efeito contrário (“*pessimização*”).
- Escolhendo os flags de otimização que devem ser usados
 - Teste os flags recomendados pelo fabricante do compilador;
 - Utilize os flags mais conhecidos e avalie as diferenças;
- **OBSERVAÇÃO CRUCIAL:**
 - Os flags de otimização podem influenciar severamente no resultado de alguns cálculos, portanto, certifique-se que o programa otimizado está produzindo o resultado correto.



Otimização assistida

- Alguns compiladores possuem opções específicas para a geração de relatórios de avaliação de desempenho;
 - Compila-se o programa com essas opções ativadas;
 - Executa-se uma rodada normal do software. Será produzido o relatório de otimização;
 - Recompila o software fornecendo o relatório de otimização para o compilador;
 - Baseado no relatório de otimização o compilador tentará produzir uma versão de código binário mais eficiente que o primeiro.
- Enquanto houver algum ganho esse ciclo pode ser repetido



Alguns fabricantes de compiladores chamam este procedimento de PGO **P**rofile **G**uided **O**ptimization



Otimização assistida (cont.)

- Fase 1:

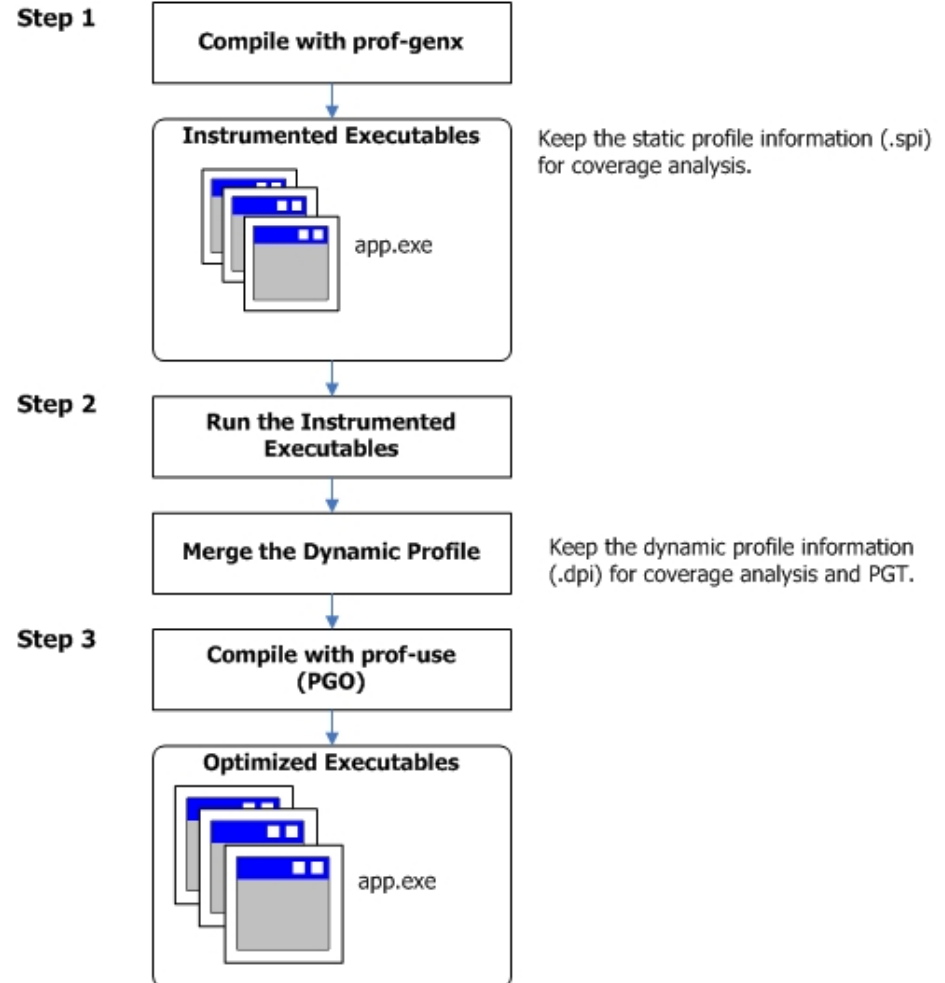
```
ifort -prof-gen -c foo.f  
ifort -o foo foo.o
```

- Fase 2:

```
./foo
```

- Fase 3:

```
ifort -prof-use -c foo.f  
ifort -o foo foo.o
```





Flags Compilador

Funcionalidade	Linux
Desabilita otimizações	-O0
Otimização (não aumenta tamanho do código)	-O1
Otimização (habilita vetorização, otimizações de laços)	-O2
Otimização alta-ordem (loop unrol)	-O3
Vectorization	<várias Opções>
Otimização Agressiva (-ipo, -O3, -no-prec-div, -xHost)	-fast
Criação de informações para depuração	-g
Gera arquivo Assembly	-S
Geração do relatório de otimização	-qopt-report=<n>
Suporte OpenMP	-qopenmp



LAB01- Otimização Automática

- Objetivos:
 - Explorar opções de otimização do compilador a partir de uma implementação de um algoritmo de atualização vetorial do tipo: SAXPY
 - Demonstrar o ganho de desempenho com a utilização de bibliotecas especializadas.



Lab 01 – SETUP UP

- Obter dados do *handson* no site do workshop.
- Copiar os arquivos do hands-on para seu diretório home.
 - `cp handson.tar.gz ${HOME}/.`
- Descompacta-lo:
 - `tar xzvf handson.tar.gz`



LAB1: SAXPY

- Localização dos arquivos:
 - `~/handson/lab01/saxpy`
- Para compilar sem otimização: `-O0`
 - `icc -g -O0 saxpy1.cpp -o saxpy.exe`
- Compile com as opções `-O1`, `-O2` e `-O3 -xHost`
- Veja o relatório de otimização:
 - acrescente a flag `-qopt-report`



LAB01: Multiplicação Matricial

- Objetivos: Demonstrar o uso de bibliotecas eficientes
- Localização do código-fonte:
 - `~/handson/lab01`
- Para compilar:
 - Versão ingênua: `$make step1`
 - Versão usando MKL Blas: `$make step2`
- Compare os tempos de processamento.

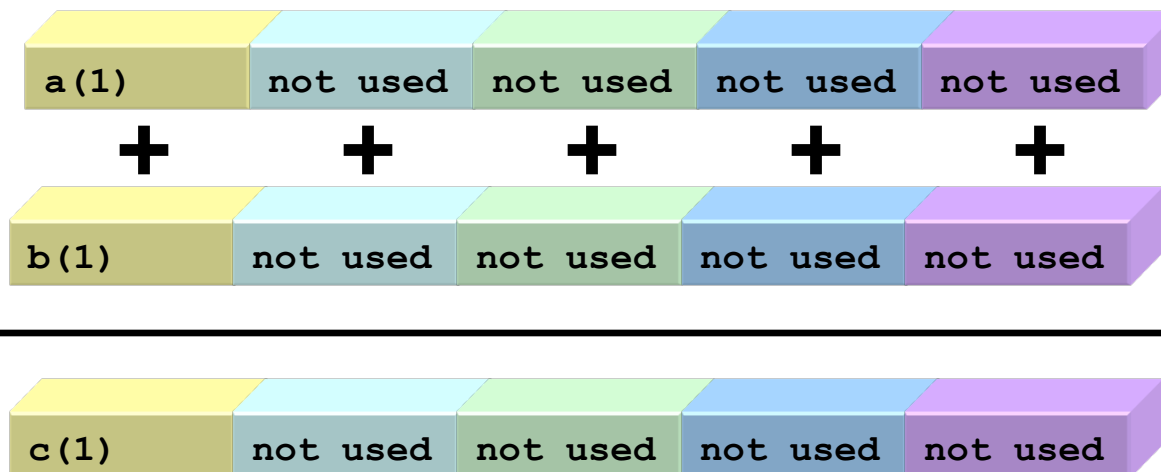
VETORIZAÇÃO



Vetorização: Como funciona?

- Uma operação **ESCALAR** pode ser ilustrada por:

```
do i=1,n  
    c(i) = a(i) + b(i)  
enddo
```

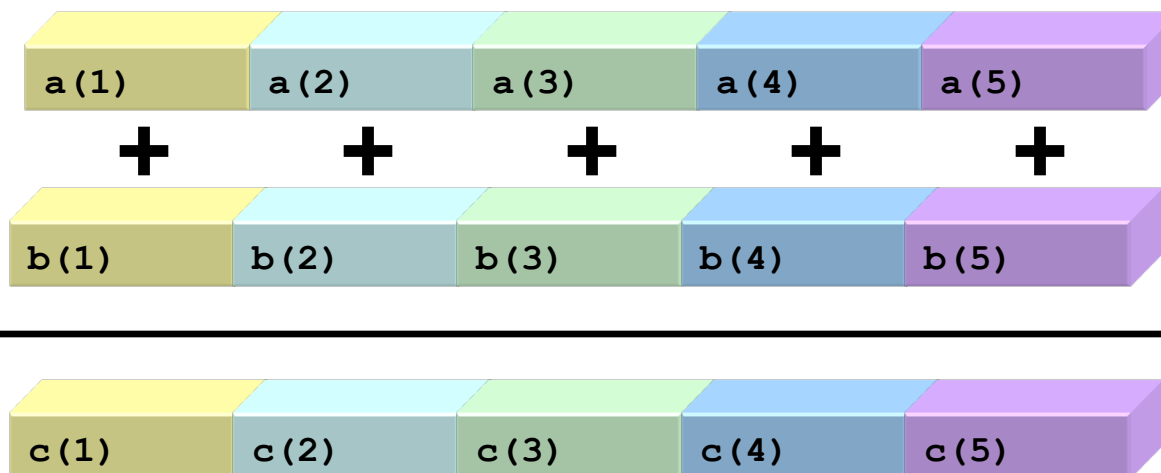




Vetorização: Como funciona? (cont.)

- Executando-se a mesma operação em um processador **VETORIAL** teríamos:

```
do i=1,n  
    c(i) = a(i) + b(i)  
enddo
```



ou seja, poderíamos operar em 5 posições dos vetores simultaneamente a cada ciclo do processador (5 x mais rápido!).



Vetorização - Perspectivas

- Perspectiva para o hardware: Instruções, registradores ou unidades funcionais especializadas que permitem o paralelismo das operações no interior do núcleo em arranjos de dados (vetores)
- Perspectiva do Compilador: determinar como e quando é possível expressar computações em termos de instruções vetoriais
- Perspectiva do usuário: determinar como escrever códigos de uma maneira que permita ao compilador deduzir que a vetorização é possível



Vetorização: Hardware

- Objetivo: paralelizar computação sobre arranjos vetoriais
- Duas principais técnicas: *pipelining* e SIMD (*Single Data Multiple Data*)
- *Pipelining*: Diversas tarefas distintas executando simultaneamente
- SIMD: Muitas instâncias de uma simples tarefa executando simultaneamente
 - Vetores menores, poucos ciclos por instrução
 - Novos CPUs podem usar o *pipelining* em algumas instruções SIMD



Pipelining: como funciona?

Clock Cycle	R1	R2	R3	R4
1	X_1			
2	X_2	X_1^2		
3	X_3	X_2^2	X_1^2+8	
4	X_4	X_3^2	X_2^2+8	$(X_1^2+8)/2$
5	X_5	X_4^2	X_3^2+8	$(X_2^2+8)/2$
6	Load	X_5^2	X_4^2+8	$(X_3^2+8)/2$
7	Square		X_5^2+8	$(X_4^2+8)/2$
8		Add		$(X_5^2+8)/2$

Divide



Vetorização e Pipelining

Clock	R1	R2	R3	R4
1	$[X_1, X_2 \dots X_5]$			
2		$[X_1^2, X_2^2 \dots X_5^2]$		
3			$[X_1^2+8, X_2^2+8 \dots X_5^2+8]$	
4	Load	Square	Add	$[(X_1^2+8)/2,$ $(X_2^2+8)/2$... $(X_5^2+8)/2]$

Divide

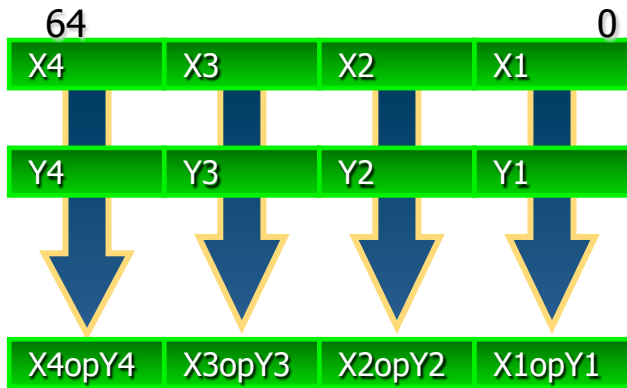


Vetorização: Motivação

- Velocidade dos CPUs alcançou um patamar
 - Limitações de consumo
 - Muitos transistores “lentos” são mais eficiente que poucos transistores “rápidos”
- Melhorias de processo ocasionou barateamento do espaço físico
 - Uma solução: Adicionar mais cores
 - Primeiro Intel dual cores CPUs aparece em 2005.
 - Número cresce rapidamente (+60 no MIC)
 - Outra solução: + FPU por core – Operações vetoriais
 - Surgimento no Pentium com MMX em 1996
 - Rápido aumento da largura do vetor (512-bit [8 doubles]) no MIC



Evolução do processamento vetorial dos processadores Intel®



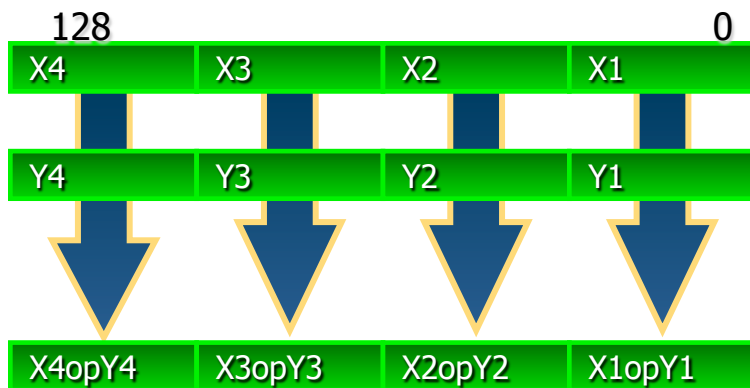
MMX™

Vector size: 64bit

Data types: 8, 16 and 32 bit integers

VL: 2,4,8

For sample on the left: X_i, Y_i 16 bit integers



Intel® SSE

Vector size: 128bit

Data types:

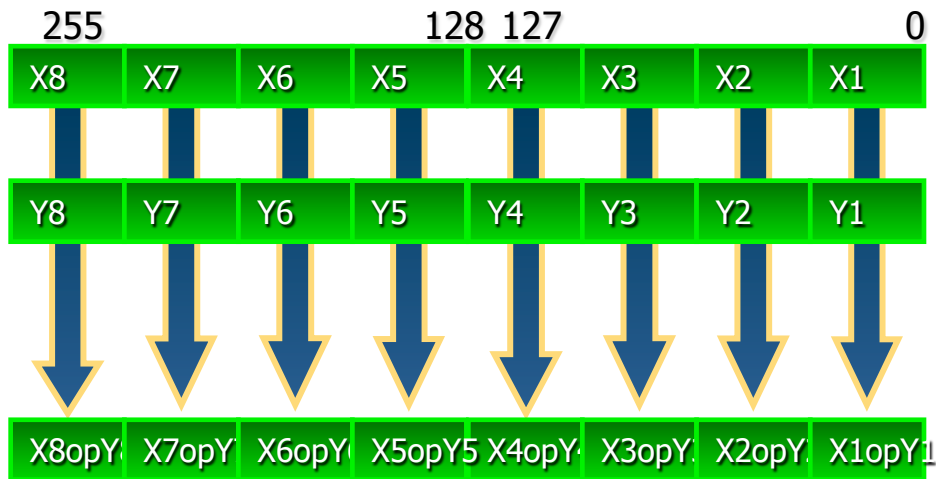
8,16,32,64 bit integers

32 and 64bit floats

VL: 2,4,8,16

Sample: X_i, Y_i bit 32 int / float

Evolução do processamento vetorial dos processadores Intel®



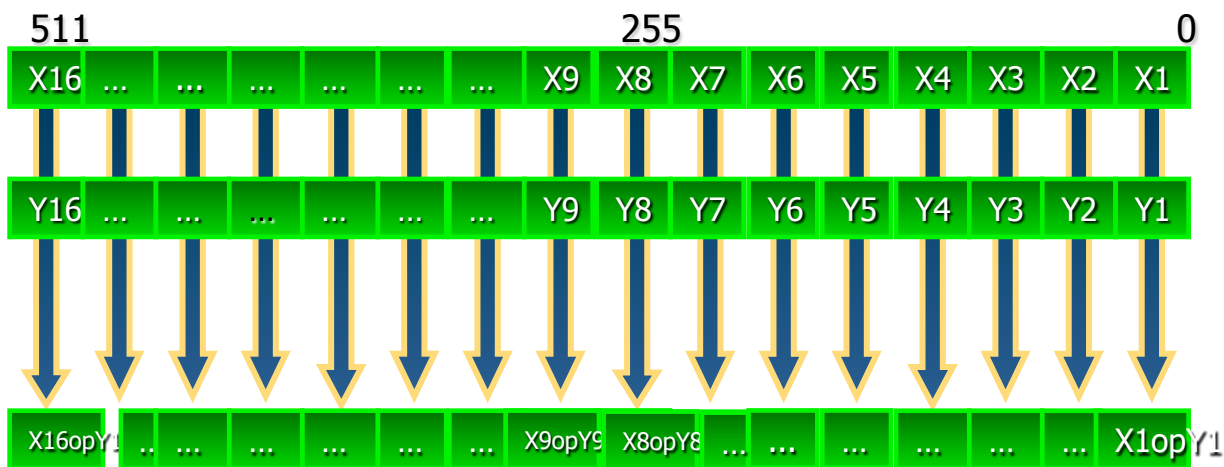
Intel® AVX / AVX2

Vector size: 256bit

Data types: 32 and 64 bit floats

VL: 4, 8, 16

Sample: X_i, Y_i 32 bit int or float



Intel® MIC / AVX-512

Vector size: 512bit

Data types:

32 and 64 bit integers

32 and 64bit floats

(some support for 16 bits floats)

VL: 8,16

Sample: 32 bit float



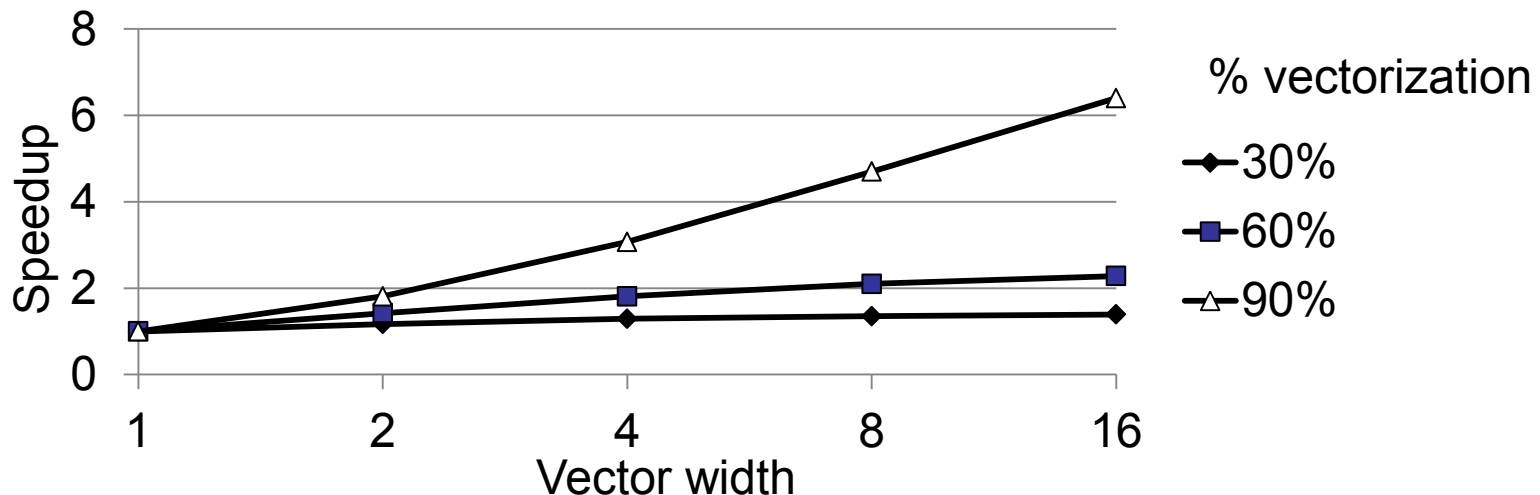
Speedup

- Paralelismo SIMD – tipicamente 1 ciclo por operação com ponto-flutuante
 - Exceções: divisões, sqrt, sin
- Speedup (comparado com código sem vetorização)
 - 128-bit SSE – 2x double, 4x single
 - 256-bit AVX – 4x double, 8x single
 - 512-bit AVX2 – 8x double, 16x single
- Exemplo AVX hipotético: 8 cores/CPU*4 double/vetor *
2.0 GHz = 64 Gflops/CPU
 - Pipelining pode melhorar ainda mais!



Speedup

- Largura de banda da memória pode ser um problema, vamos explorar isso mais tarde
 - Utilização ineficiente da cache, alinhamento, latência
- SIMD é paralelo → lei de Amdahls é efetivo!
 - Porções seriais/escalares do código ou CPU são limitadores
 - *Speedup* teórico é somente um limitante superior.



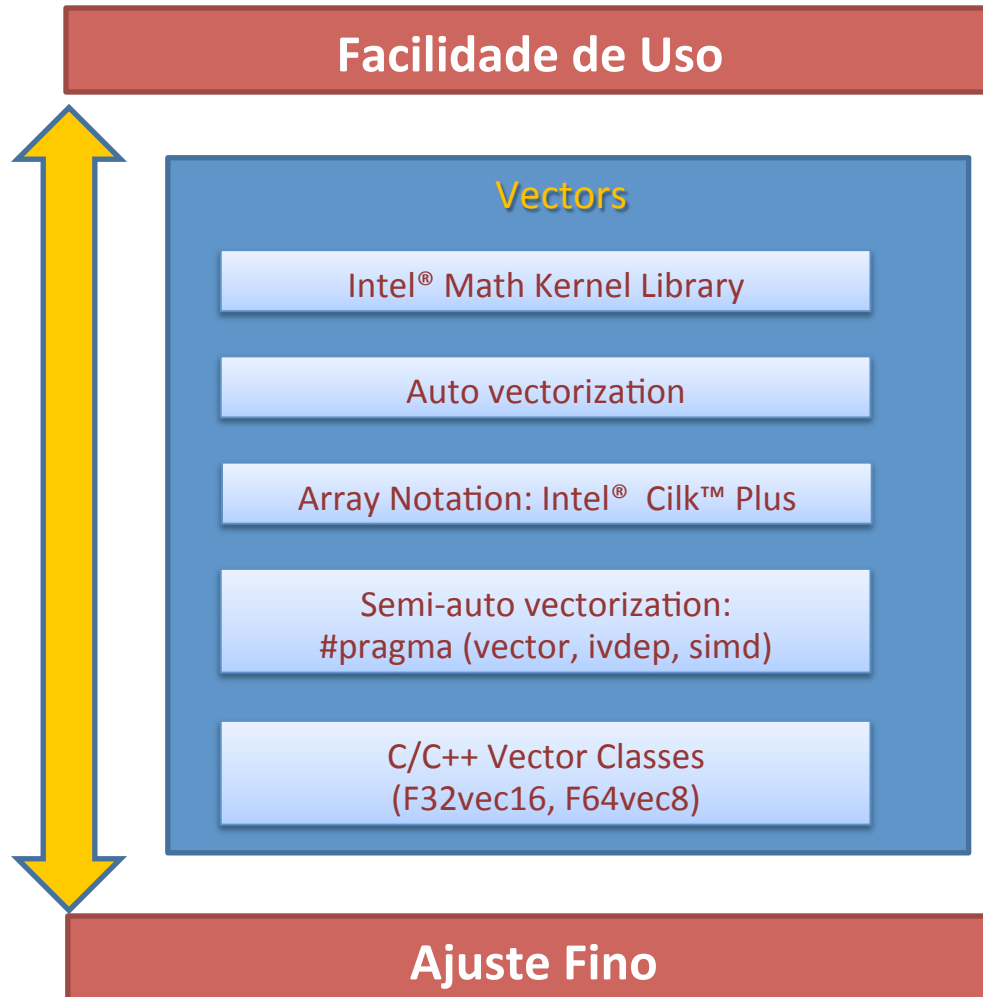


Perspectiva do Usuário

- Como podemos aproveitar esse poder?
 - Codificação *Assembly*
 - Potencial ganho de desempenho final mas somente para os bravos
 - Funções Intrínsecas
 - Passo anterior ao *assembly*. Útil mas arriscado
 - Deixe o compilador descobrir como... (auto-vetorização)
 - Relativamente simples, desafiador para o compilador
 - O compilador pode necessitar de uma ajuda
 - Link com um biblioteca otimizada que faça o trabalho real
 - Exemplo: Intel MKL, escrito por pessoas que sabem todos os truques
 - Obtém os benefícios quando executados em plataformas suportadas

Identificando oportunidades de otimização

Maneiras de vetorizar o código



- Devemos avaliar três fatores:
 - Necessidade de performance
 - Disponibilidade de recursos para otimizar o código
 - Portabilidade do código



O Que eu preciso fazer?

- Conhecer o que faz ser “vetorizável”
 - Laços “for” em C/C++ ou “do” em Fortran que atendam a certas restrições
- Conhecer onde a vetorização irá ajudar
- Avaliar a saída do compilador
 - Ele está realmente vetorizado onde eu acho que deveria estar?
- Avaliar o desempenho da execução
 - Comparar com o *speedup* teórico
- Conhecer o padrão de acesso ao dados para maximizar a eficiência
- Implementar correções: diretivas, *flags* compilador e alteração do código
 - Remover construções que fazem vetorização impossível
 - Forçar vetorização quando compilador não faz, mas que deveria
 - Melhorar o padrão de acesso a memória.



Escrevendo laços “vetorizáveis”

- Requerimentos básicos:
 - Contável em tempo de execução
 - Número de iterações conhecido antes de sua execução
 - Sem terminação condicional (sem breaks)
 - Ter um fluxo de execução simples
 - Sem saltos condicionais
 - If são permitidos quando eles podem ser implementados com máscaras
 - Deve ser o laço mais interno se houver laços aninhados
 - Compilador pode reverter a ordem
 - Evitar chamadas de função
 - Funções básicas matemáticas são permitidas: *pow*, *sqrt*, *sin*
 - Algumas funções inline são permitidas



Auto-vetorização

- Suporte compilador Intel:
 - Vetorização é realizado a partir do nível de otimização `-O2`
 - Default para instruções SSE
 - Pode incorporar instruções SSE e AVX em um mesmo binário com a flag `-axAVX`
 - Rodará AVX em CPUs com AVX suporte, SSE caso contrário.
 - `-qopt-report=<n>` para gerar um relatório de vetorização.
 - AVX512: Suporte para versões `> 15.0`.
- GCC (`>4.9`):
 - Vetorização é desabilitado por default
 - Use a flag `-ftree-vectorize` combinado com nível otimização `> -O2`
 - Padrão instruções SSE, `-mavx -march=corei7-avx` para AVX
 - `-ftree-vectorizer-verbose` para gerar o relatório.



Desafio

Dependência de dados

- Vetorização pode alterar a ordem da computação em comparação de um caso escalar
- Compilador deve assegurar que a vetorização irá produzir resultados corretos
- Necessidade de considerar operações independentes em laços desenrolados – depende do tamanho do vetor
- Compilador executa uma análise de dependência



Dependências: leitura após escrita

Considere o laço:

$a = \{0, 1, 2, 3, 4\}$

$b = \{5, 6, 7, 8, 9\}$

```
for(i=1; i < N; i++)  
    a[i] = a[i-1] + b[i]
```

Aplicando cada operação sequencialmente:

$$a[1] = a[0] + b[1] \rightarrow a[1] = 0 + 6 \rightarrow a[1] = 6$$

$$a[2] = a[1] + b[2] \rightarrow a[2] = 6 + 7 \rightarrow a[2] = 13$$

$$a[3] = a[2] + b[3] \rightarrow a[3] = 13 + 8 \rightarrow a[3] = 21$$

$$a[4] = a[3] + b[4] \rightarrow a[4] = 21 + 9 \rightarrow a[4] = 30$$

$a = \{0, 6, 13, 21, 30\}$



Dependências: Leitura após escrita

Vamos tentar vetorizar:

$a = \{0,1,2,3,4\}$

$b = \{5,6,7,8,9\}$

```
for(i=1; i < N; i++)  
    a[i] = a[i-1] + b[i]
```

Aplicando operação vetorial, $i=\{1,2,3,4\}$:

$a[i-1] = \{0,1,2,3\}$ (load)

$b[i] = \{6,7,8,9\}$ (load)

$\{0,1,2,3\} + \{6,7,8,9\} = \{6,8,10,12\}$ (operate)

$a[i] = \{6,8,10,12\}$ (store)

$a = \{0,6,8,10,12\} \neq \{0,6,13,21,30\}$

NÃO VETORIZADO



Dependências: escrita após leitura

Vamos tentar vetorizar:

$a = \{0, 1, 2, 3, 4\}$

$b = \{5, 6, 7, 8, 9\}$

```
for(i=0; i < N; i++)  
    a[i] = a[i+1] + b[i]
```

Aplicando cada operação sequencialmente:

$$A[0] = a[1] + b[0] \rightarrow a[0] = 1 + 5 \rightarrow a[1] = 6$$

$$A[1] = a[2] + b[1] \rightarrow a[1] = 2 + 6 \rightarrow a[2] = 8$$

$$A[2] = a[3] + b[2] \rightarrow a[2] = 3 + 7 \rightarrow a[3] = 10$$

$$A[3] = a[4] + b[3] \rightarrow a[3] = 4 + 8 \rightarrow a[4] = 12$$

$$a = \{6, 8, 10, 12, 4\}$$



Dependências: Leitura após escrita

Vamos tentar vetorizar:

$a = \{0,1,2,3,4\}$

$b = \{5,6,7,8,9\}$

```
for (i=0; i < N; i++)  
    a[i] = a[i+1] + b[i]
```

Aplicando operação vetorial, $i=\{1,2,3,4\}$:

$a[i-1] = \{1,2,3,4\}$ (load)

$b[i] = \{5,6,7,8\}$ (load)

$\{1,2,3,4\} + \{5,6,7,8\} = \{6,8,10,12\}$ (operate)

$a[i] = \{6,8,10,12\}$ (store)

$q = \{0,6,8,10,12\} == \{0,6,8,10,12\}$

VETORIZADO



Tipos de Dependências

- Leitura após escrita:
 - Também conhecida como dependência de fluxo
 - Variável escrita primeiro e depois lida
 - Não vetorizável

```
for(i=1; i < N; i++)  
    a[i] = a[i-1] + b[i]
```

- Escrita seguida por leitura
 - Conhecida como anti dependência
 - Variável lida primeiro, depois escrita
 - Vetorizável

```
for(i=0; i < N; i++)  
    a[i] = a[i+1] + b[i]
```



Tipos de Dependências

- Leitura após leitura:
 - Na verdade não é um dependência
 - vetorizável

```
for(i=0;i < N; i++)  
    a[i] = b[i%2] + c[i]
```

- Escrita após escrita
 - Dependência de saída
 - Variável escrita e reescrita
 - Não vetorizável

```
for(i=0;i < N; i++)  
    a[i%2] = b[i] + c[i]
```



Dependência em Laços: “Aliasing”

- Em C, ponteiros podem esconder dependências de dados
 - Pode haver sobreposição de memória
- O código abaixo é seguro?

```
void compute(double *a, double *b,  
            double *c)  
{  
    for(i =1; i < N; i++) {  
        a[i] = b[i] + c[i];  
    }  
}
```

Não se forem dados os
argumentos: compute(a, a+1,c)



Dependência em Laços: Aliasing

- C99 introduz a palavra chave *restrict* na linguagem
 - Diz para o compilador assumir que os endereços não irão sobrepor.
 - Pode ser necessário usar as *flags*: `-restrict`, `-std=c99`

```
void compute(double * restrict a,  
             double * restrict b,  
             double * restrict c)  
{  
    for(i =1; i < N; i++) {  
        a[i] = b[i] + c[i];  
    }  
}
```



Relatório de Vetorização

- Mostra quais laços foram ou não vetorizados, e porque
- Intel: `-qopt-report=<n>`
 - 0: nenhuma informação
 - 1: lista laços vetorizados
 - 2: lista laços não vetorizados, com explicação
 - 3: saída adicional com informações de dependência
 - 4: lista laços não vetorizados, sem explicação
 - 5: lista laços não vetorizados, com informação de dependência
- Relatórios são essenciais para determinar onde o compilador encontrou uma dependência
- Compilador é conservador. Você precisa ir ao código e verificar se realmente há uma dependência.



Sugestões para Vetorização

- Compilador deve comprovar que não há nenhuma dependência de dados que irá afetar a exatidão do resultado
- Entretanto, as vezes isso é impossível
 - Ex. Uso complicado dos ponteiros
- Solução dos compiladores Intel: diretiva IVDEP (Ignore Vector DEPENDence)
 - Diz ao compilador: assume nenhuma dependência

```
subroutine
vec1(s1,M,N,x)
...
!DEC$ IVDEP
do i = 1,N
  x(i) = x(i+M) + s1
end do
```

```
void vec1(double s1,int M,
          int N,double *x) {
...
#pragma IVDEP
for(i=0;i<N;i++) x[i]=x[i+M]+s1;
```




Diretivas que auxiliam a vetorização

- Antes do laço: `#pragma (C/C++) !DIR$` (para Fortran)
- `ivdep`
 - Instrui o compilador a ignorar as dependências assumidas
 - Compilador pode vetorizar laços que ele inicialmente considerou não vetorizável
- `vector always`
 - Sobrescreve as heurísticas do compilador que determinam se a vetorização é susceptível de produzir ganhos de desempenho;
- `novector`
 - Não vetorize
- `omp simd` (OpenMP 4.0)
 - Vetoriza laço (mais poderoso que `vector always`)



Exemplos (C/C++)

```
void ignore_vec_dep(int *a, int k, int c, int m) {  
    #pragma ivdep  
    for (int i = 0; i < m; i++)  
        a[i] = a[i + k] * c;  
}
```

```
void vec(int *a, int *b, int m) {  
    #pragma vector always  
    for(int i = 0; i <= m; i++)  
        a[32*i] = b[99*i];  
}
```

```
#pragma omp simd collapse(2)  
for(i=0; i<N; i++) { a[i] = b[i] * c[i];  
    for(i=0; i<N; i++) { d[i] = e[i] * f[i]; }  
}
```



Exemplos (Fortran 90)

```
!dir$ ivdep
```

```
do i = 1, m;
```

```
    a(i) = a(i + k) * c;
```

```
enddo
```

```
!dir$ vector always
```

```
do i = 1, m
```

```
    a(32*i) = b(99*i)
```

```
enddo
```

```
!$omp simd collapse(2)
```

```
do j=1, N
```

```
    do i=1, N
```

```
        a(i,j) = a(i,j) * b[i]
```

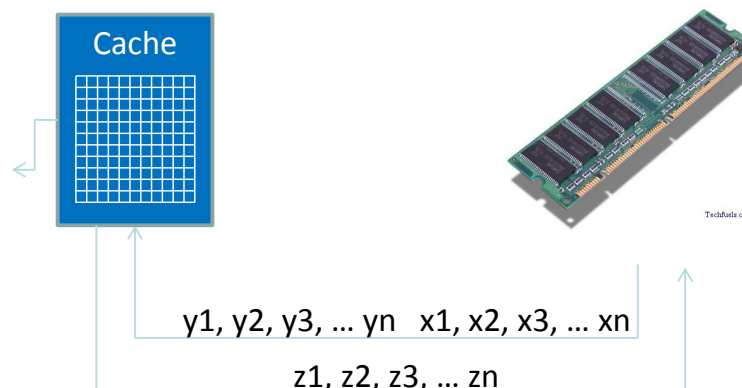
```
    enddo
```

```
enddo
```



CACHE e Alinhamento

$$\begin{matrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{bmatrix} \\ \text{ymm2} \end{matrix} = a * \begin{matrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \\ \text{ymm0} \end{matrix} + \begin{matrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \\ \text{ymm1} \end{matrix}$$



- Vetorização eficiente requer preocupações que vão além da unidade SIMD
 - Registradores: alinhamento de dados nos limites de 128, 256 e 512 bits
 - Cache: Cache é rápida, memória é lenta
- Memória: acesso sequencial muito mais rápido que acesso randômico ou espaçado.
- Se as unidades vetoriais ficam esperando dados, a eficiência é drasticamente prejudicada.



Acesso com passo largo

- Padrão de acesso mais rápido é passo unitário
- Melhor desempenho quando CPU pode carregar L1 Cache da memória em blocos de dados, sequencialmente.
- Construções Passo Unitário:
 - arranjos multidimensional
 - Fortran: passo unitário na dimensão mais interna
 - C/C++: passo unitário na dimensão mais externa

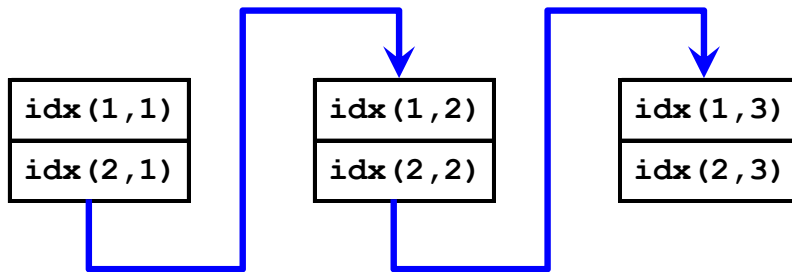
```
do j = 1,n; do i=1,n  
  a(i,j)=b(i,j)*s  
enddo; endo
```

```
for (j=0;j<n;j++)  
  for (i=0;i<n;i++)  
    a[j][i]=b[j][i]*s;
```



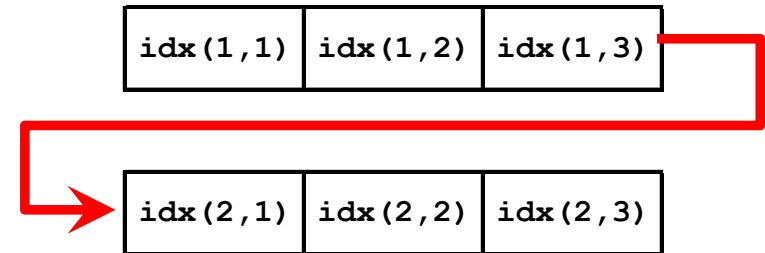
Ordem de armazenamento de dados em arranjos

Fortran (por colunas)



`integer :: idx(2,3) = (/1,1,2,2,3,3/)`

C/C++ (por linhas)



`integer :: idx(2,3) = (/1,2,3,1,2,3/)`

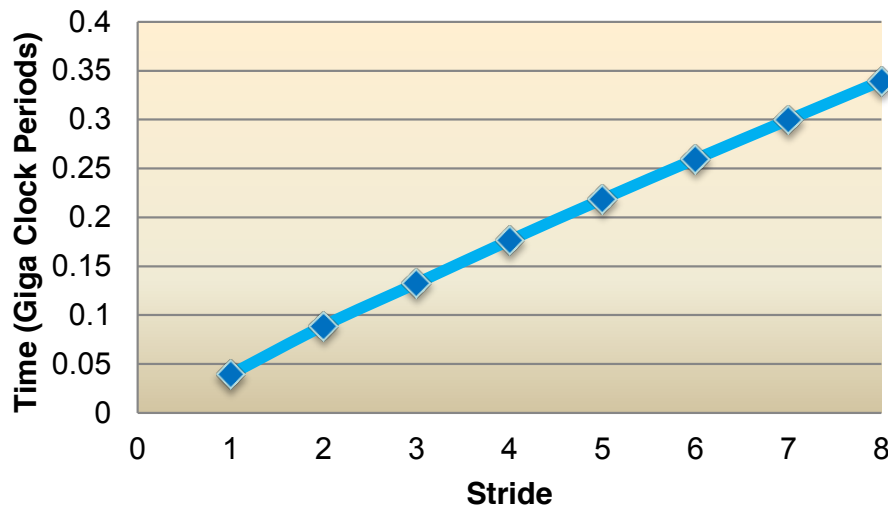
$$\mathbf{idx} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$



Acesso com passo largo

- Acesso espaçado na memória reduz a largura de banda efetiva da memória.
- Pior que acesso não alinhado. Muitas operações para povoar uma *cache line*, registradores, etc.

Memory Strided Add* Performance



```
*do i = 1, 4000000*istride, istride  
    a(i) = b(i) + c(i) * sfactor  
enddo
```



Blocagem

Funciona com pequenos blocos de matrizes quando a expressão contém acessos a memória com espaçamentos mistos.

```
do i=1,n
  do j=1,n
    A(j,i)=B(i,j)
  end do
end do
```



```
do i=1,n,2
  do j=1,n,2
    A(j ,i )=B(i ,j )
    A(j+1,i )=B(i+1,j )
    A(j ,i+1)=B(i ,j+1)
    A(j+1,i+1)=B(i+1,j+1)
  end do
end do
```



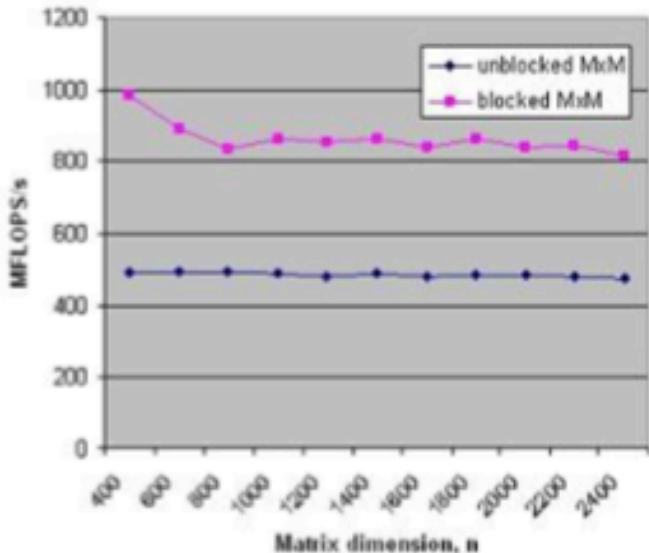

Blocagem

matrix
multiplication
example

```

real*8 a(n,n) , b(n,n) , c(n,n)
do ii=1,n,nb
  do jj=1,n,nb
    do kk=1,n,nb
      do i=ii,min(n,ii+nb-1)
        do j=jj,min(n,jj+nb-1)
          do k=kk,min(n,kk+nb-1)
            c(i,j)=c(i,j)+a(j,k)*b(k,i)

```

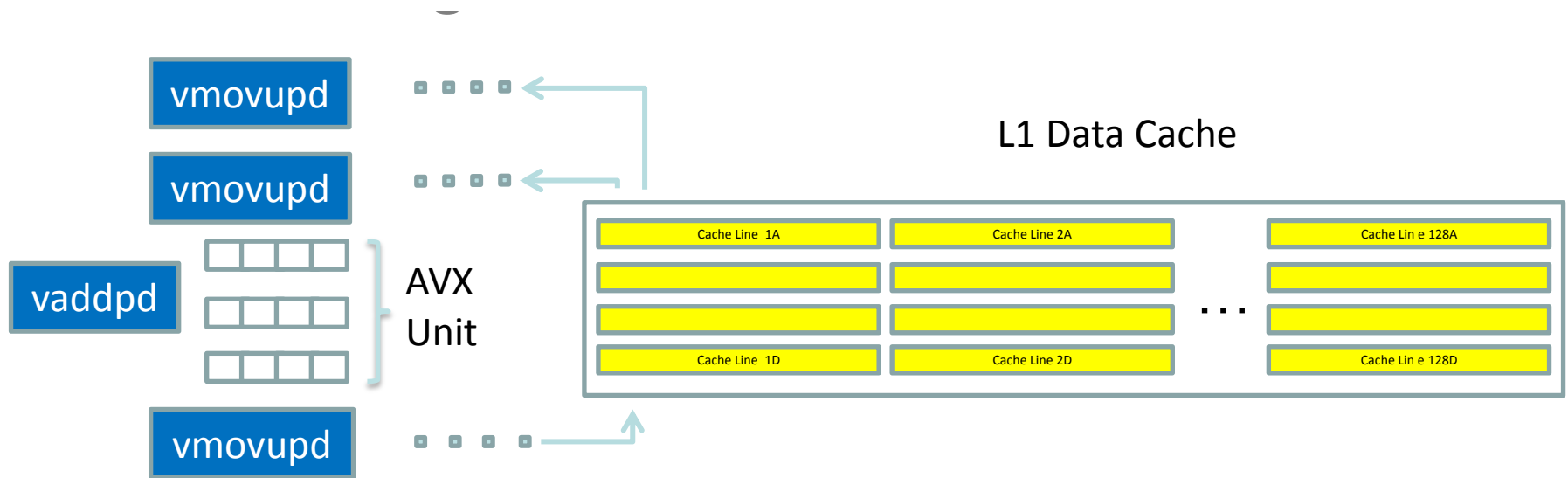


```

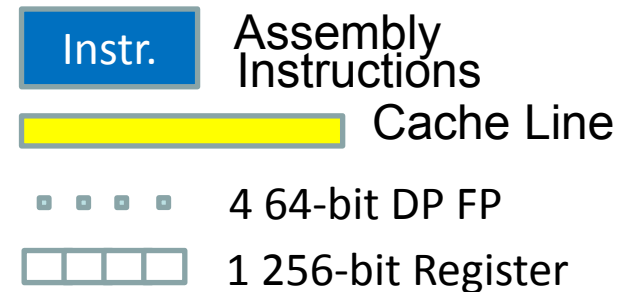
end do; end do; end do; end do; end do; end do

```

Alinhamento dos dados



- Instruções de carregamento movem múltiplos valores da cache para os registradores simultaneamente.
- Mais rápido quando a *cache line* inteira é movida com um única unidade, i.e., alinhada



Alinhamento

Alignment

32-byte (AVX)
aligned



- Load 4 DP Words
- Load 4 DP Words
- Load 4 DP Words
- Load 4 DP Words

registers

Non-aligned



- Load 4 DP Words
- Load 4 DP Words
- Load 4 DP Words
- Load 4 DP Words

registers



Alinhamento

- Aplicado especialmente a arranjos, estruturas
 - Iterar arranjos multidimensionais podem afetar o alinhamento se número de colunas/linhas não forem múltiplos do tamanho da *cache line*.
 - Solução: usar *padding* e adaptar seu algoritmo
- Alinhamento depende da arquitetura do processador:
 - Haswell: 64 bytes
 - MIC: 64 bytes



Alinhamento Manual

- Para Intel, diretivas podem forçar o compilador a assumir alinhamento
 - **#pragma vector aligned (c/c++)** afirma que dados em um laço estão alinhados para os limites apropriados.
 - Fortran 90: **!dir\$ vector aligned**
 - Cuidado – *segfault* se você estiver errado

```
// C/C++  
#pragma vector aligned  
for(i=0;i<500;i++)  
    B[i] = A(i,3)
```

```
! FORTRAN  
!dir$ vector aligned  
do i=1,500  
    B(i) = A(3,i)  
end do
```

- Pode forçar alocação dinâmica ser alinhada
 - Com compiladores Intel, use **_mm_malloc** ou **_mm_free**



Alinhamento

- Definindo alinhamento de arranjos estáticos:

- Windows C/C++:

- `__declspec(align(64)) float A[1000];`

- Linux C/C++:

- `float A[1000] __attribute__((aligned(64)));`

- Fortran:

- `Real :: A(1000)`

- `!dir$ attributes align: 64::A`

- Informando o compilador sobre o alinhamento:

```
// C/C++  
void myfunc (double p[]){  
    __assume_aligned(p, 64);  
    for(int i=0; i < n; i++)  
        p[i]++;  
}
```

```
!Fortran  
!dir$ assume_aligned A:64  
DO I = 1, N  
    A(i) = A(i) + 1  
END DO
```



Diagnosticando deficiências da cache e memória

- Maus padrões de acesso pode impedir vetorização
 - No relatório de vetorização: “*vectorization possible but seems inefficient*”
- Caso contrário, pode ser de difícil detecção
 - Indicador: desempenho da vetorização longe do esperado
- Ferramentas de perfilagem podem ajudar:
 - Intel Vtune: Microarchitecture Analysis / Bandwidth
 - Visualizar o número de ciclos do CPU gasto com acesso aos dados (L1 cache miss, TLB misses, etc)



Vetorização

Resumo

- Vetorização ocorre em laços e podem ser feitas “automaticamente” pelo compilador
- Precisa saber onde a vetorização pode ocorrer e verificar se o compilador está realmente fazendo.
- Precisa saber se a falha do compilador na vetorização é legítima
 - Se for, corrigir o código, se não, usar `#pragma`.
- Ter consciência dos efeitos relacionadas a forma como os dados são acessados e o alinhamento deles.
 - As unidades vetoriais precisam ser bem alimentadas!



LAB #2

- **Objetivos: explorar as possibilidades da vetorização**
 - **Dois código-fonte:**
 - **Multiply: Multiplicação Matricial**
 - Código base: multiply0.c
 - Arquivos: handson/lab02/multiply
 - **Sten2D: Stencil de diferença finitas 2D**
 - Código base: stenc2d_base.c
 - Arquivos: handson/lab02/stend2d
 - **Exemplos em Fortran:**
 - Handson/lab02/Fortran



LAB02: Multiply

- Step 1: Compile multiply0.c com `-O3 -qopt-report=3` e veja o relatório gerado.

```
LOOP BEGIN at multiply0.c(47,5) inlined into multiply0.c(106,2)
remark #25444: Loopnest Interchanged: ( 1 2 3 ) --> ( 1 3 2 )
remark #15542: loop was not vectorized: inner loop was already vectorized

LOOP BEGIN at multiply0.c(49,10) inlined into multiply0.c(106,2)
remark #15542: loop was not vectorized: inner loop was already vectorized

LOOP BEGIN at multiply0.c(48,9) inlined into multiply0.c(106,2)
<Peeled loop for vectorization>
LOOP END

....
LOOP BEGIN at multiply0.c(48,9) inlined into multiply0.c(106,2)
<Remainder loop for vectorization>
remark #15335: remainder loop was not vectorized: vectorization possible but seems inefficient.
Use vector always directive or -vec-threshold0 to override
```

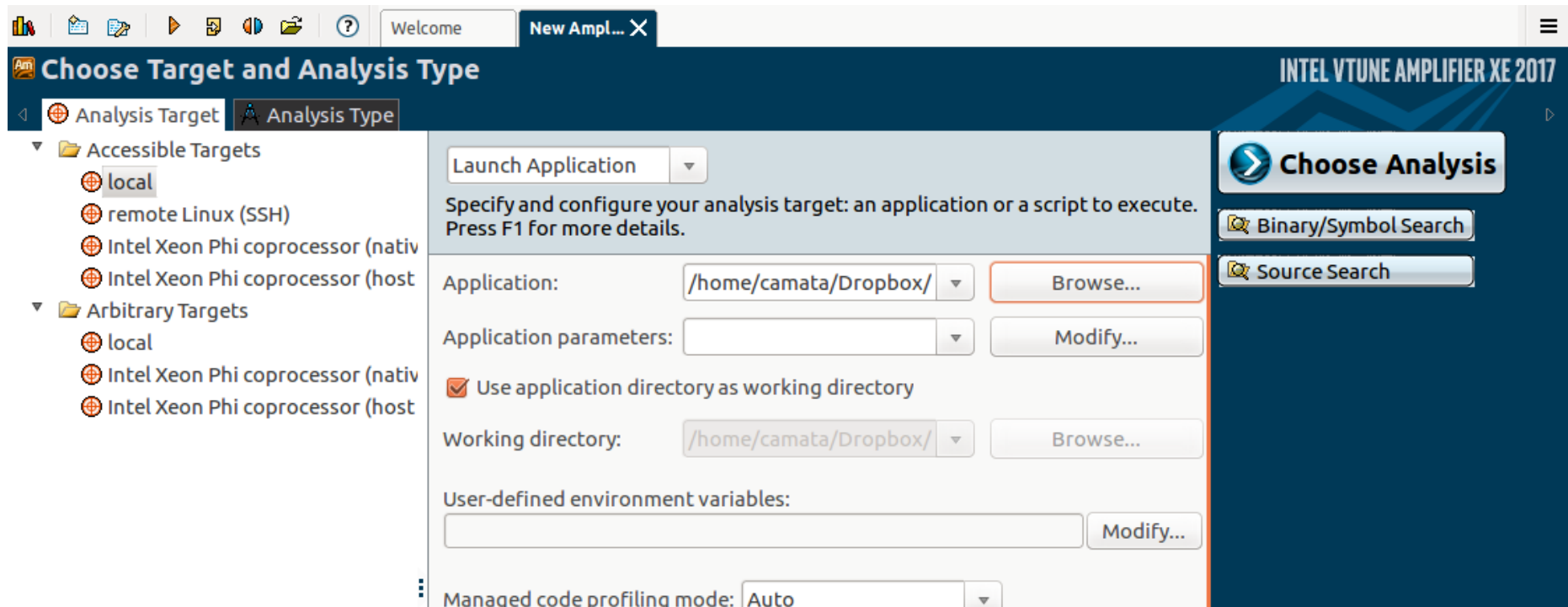


Vtune: Coletando dados de desempenho

- Processo em três etapas:
 - Compile com a opção `-g`
 - Execute `amplxe-cl` para coletar os dados de perfilagem
 - Visualize os dados com `amplxe-gui`
- O que coletar:
 - `amplxe-cl -help collect`
- Identificando gargalos:
 - `amplxe-cl -collect hotspots ./myexe`
- Visualizando em TXT
 - `amplxe-cl -report summary -r ./r000hs`

Perfilando com o vtune

- Criando o projeto de perfilagem local



The screenshot shows the Intel VTune Amplifier XE 2017 interface. The main window is titled "Choose Target and Analysis Type". On the left, there is a tree view with two categories: "Accessible Targets" and "Arbitrary Targets". Under "Accessible Targets", there are options for "local", "remote Linux (SSH)", and "Intel Xeon Phi coprocessor (native)" and "(host)". Under "Arbitrary Targets", there are options for "local", "Intel Xeon Phi coprocessor (native)", and "(host)". The main area of the dialog is for configuring the analysis target. It has a "Launch Application" dropdown menu. Below it, a text box says "Specify and configure your analysis target: an application or a script to execute. Press F1 for more details." There are several input fields: "Application:" with a dropdown menu showing "/home/camata/Dropbox/" and a "Browse..." button; "Application parameters:" with a dropdown menu and a "Modify..." button; "Working directory:" with a dropdown menu showing "/home/camata/Dropbox/" and a "Browse..." button; and "User-defined environment variables:" with a text box and a "Modify..." button. There is also a checkbox labeled "Use application directory as working directory" which is checked. At the bottom, there is a "Managed code profiling mode:" dropdown menu set to "Auto". On the right side of the dialog, there is a "Choose Analysis" button and two search options: "Binary/Symbol Search" and "Source Search".



Vtune Amplifier XE 2017

The screenshot shows the Intel VTune Amplifier XE 2017 interface. The main view is 'General Exploration' for a process named 'r000ge'. The 'Elapsed Time' is 4.708s. The interface displays various performance metrics, with 'Memory Bound' being the most significant bottleneck at 69.9%.

Metric	Value
Elapsed Time	4.708s
Clockticks	12,095,500,000
Instructions Retired	22,752,800,000
CPI Rate	0.532
MUX Reliability	0.773
Front-End Bound	0.2%
Bad Speculation	0.0%
Back-End Bound	87.8%
Memory Bound	69.9%
L1 Bound	0.082%
L3 Bound	
Contested Accesses	0.000
Data Sharing	0.000
LLC Hit	0.716%
SQ Full	0.009
DRAM Bound	
Memory Latency	
LLC Miss	0.028
Store Bound	0.000
Core Bound	18.0%
Retiring	24.0%
Total Thread Count	1
Paused Time	0s

At the bottom of the screenshot, there is a section for 'CPU Usage Histogram'.

- Tipos de Análises:
1. Basic Hotspots (CPU TIME)
 2. Advanced Hotspots (CPU TIME, CALL STACK,...)
 3. General-Exploration (Microarchitecture Analysis)
 4. Bandwidth
 5. CPU Specific Analysis



Bottom-UP View

General Exploration General Exploration viewpoint (change) ? INTEL VTUNE AMPLIFIER XE 2017

Analysis Target Analysis Type Collection Log Summary Bottom-up Event Count Platform multiply0.c

Source Assembly Assembly grouping: Address

S. Li.	Source	Cl.	Address	So.. Line	Assembly	Cl.
40	}		0x4014d9	50	nopl %eax, (%rax)	10,..
41			0x4014e0		Block 23:	
42	void multiply0(int msize, TYPE a[][NUM], TYPE b[][0x4014e0	50	movupsx (%r15,%rcx,8), %xmm1	270..
43	{		0x4014e5	50	movupsx 0x10(%r15,%rcx,8), %xmm2	2,2..
44	int i,j,k;		0x4014eb	50	movupsx 0x20(%r15,%rcx,8), %xmm3	438..
45			0x4014f1	50	movupsx 0x30(%r15,%rcx,8), %xmm4	595..
46	// Basic serial implementation		0x4014f7	50	mulpd %xmm0, %xmm1	202..
47	for(i=0; i<msize; i++) {	3,4..	0x4014fb	50	mulpd %xmm0, %xmm2	1,1..
48	for(j=0; j<msize; j++) {	1,0..	0x4014ff	50	mulpd %xmm0, %xmm3	788..
49	for(k=0; k<msize; k++) {	88,..	0x401503	50	mulpd %xmm0, %xmm4	938..
50	c[i][j] = c[i][j] + a[i][k] *	10,..	0x401507	50	addpdx (%r11,%rcx,8), %xmm1	348..
51	}		0x40150d	50	addpdx 0x10(%r11,%rcx,8), %xmm2	749..
			0x401514	50	addpdx 0x20(%r11,%rcx,8), %xmm3	583..
			0x40151b	50	addpdx 0x30(%r11,%rcx,8), %xmm4	1,0..
			0x401522	50	movupsx %xmm1, (%r11,%rcx,8)	557..
			0x401527	50	movupsx %xmm2, 0x10(%r11,%rcx,8)	210..
			0x40152d	50	movupsx %xmm3, 0x20(%r11,%rcx,8)	198..
			0x401533	50	movupsx %xmm4, 0x30(%r11,%rcx,8)	234..
			0x401539	48	add \$0x8, %rcx	482..

Permite identificar quais tipos de instruções foram usadas na vetorização (SSE, AVX). Nesse caso, não houve vetorização. As operações são escalares



LAB02: Multiply

- Step 2: Permute os loops e compile novamente com `-O3 -qopt-report=3` e veja o relatório gerado.

```
LOOP BEGIN at multiply0.c(48,9) inlined into multiply0.c(106,2)
```

```
<Remainder loop for vectorization>
```

```
remark #15335: remainder loop was not vectorized: vectorization possible but seems inefficient.
```

```
Use vector always directive or -vec-threshold0 to override
```

- Step 3: Alocação dinâmica usando `_mm_malloc()` e `#pragma ivdep` no loop interno



LAB02: Multiply

- Step 4: Implemente blocagem para melhorar eficiência do uso da cache.

```
mblock = MATRIX_BLOCK_SIZE;

for (i0 = 0; i0 < msize; i0 += mblock) {
    for (k0 = 0; k0 < msize; k0 += mblock) {
        for (j0 = 0; j0 < msize; j0 += mblock) {
            for (i = i0; i < i0 + mblock; i++) {
                for (k = k0; k < k0 + mblock; k++) {
#pragma ivdep
#pragma vector aligned
                    for (j = j0; j < j0 + mblock; j++) {
                        c[i][j] = c[i][j] + a[i][k] * b[k][j];
                    }
                }
            }
        }
    }
}
```




LAB02: Multiply

- Step 4: Implemente blocagem para melhorar eficiência do uso da cache.

```
LOOP BEGIN at multiply5.c(52,7)
  remark #15542: loop was not vectorized: inner loop was already vectorized
  LOOP BEGIN at multiply5.c(53,11)
    remark #15542: loop was not vectorized: inner loop was already vectorized
    LOOP BEGIN at multiply5.c(54,15)
      remark #15542: loop was not vectorized: inner loop was already vectorized
      LOOP BEGIN at multiply5.c(55,19)
        remark #15542: loop was not vectorized: inner loop was already vectorized
        LOOP BEGIN at multiply5.c(56,23)
          remark #15542: loop was not vectorized: inner loop was already vectorized
          LOOP BEGIN at multiply5.c(59,27)
            remark #15300: LOOP WAS VECTORIZED
          LOOP END
        LOOP END
      LOOP END
    LOOP END
  LOOP END
LOOP END
```



Vtune Bottom-up

General Exploration General Exploration viewpoint (change) INTEL VTUNE AMPLIFIER XE 2017

Analysis Target Analysis Type Collection Log Summary Bottom-up Event Count Platform multiply5.c

Source Assembly Assembly grouping: Address

S. Li.	Source	Cl.	Address	So.. Line	Assembly	Cl.
50	mblock = MATRIX_BLOCK_SIZE;		0x400ead	60	vmovupdy 0x80(%r14), %ymm5	294..
51			0x400eb6	60	vmovupdy 0xa0(%r14), %ymm6	584..
52	for (i0 = 0; i0 < msize; i0 +=mblock) {		0x400ebf	60	vmovupdy 0xc0(%r14), %ymm7	261..
53	for (k0 = 0; k0 < msize; k0 += mblock) {		0x400ec8	60	vmovupdy 0xe0(%r14), %ymm8	527..
54	for (j0 =0; j0 < msize; j0 += mblock	0	0x400ed1	60	vfmadd213pdy (%r12,%r15,8), %ymm0, %ymm1	197..
55	for (i = i0; i < i0 + mblock; i+	0	0x400ed7	60	vfmadd213pdy 0x20(%r12,%r15,8), %ymm0,	367..
56	for (k = k0; k < k0 + mblock	78,..	0x400ede	60	vfmadd213pdy 0x40(%r12,%r15,8), %ymm0,	246..
57	#pragma ivdep		0x400ee5	60	vfmadd213pdy 0x60(%r12,%r15,8), %ymm0,	477..
58	#pragma vector aligned		0x400eec	60	vfmadd213pdy 0x80(%r12,%r15,8), %ymm0,	215..
59	for (j = j0; j < j0 + mb	304..	0x400ef6	60	vfmadd213pdy 0xa0(%r12,%r15,8), %ymm0,	302..
60	c[i][j] = c[i][j] +	6,6..	0x400f00	60	vfmadd213pdy 0xc0(%r12,%r15,8), %ymm0,	171..
61	}		0x400f0a	60	vfmadd213pdy 0xe0(%r12,%r15,8), %ymm0,	302..
62	}		0x400f14	60	vmovupdy %ymm1, (%r12,%r15,8)	171..
63	}		0x400f1a	60	vmovupdy %ymm2, 0x20(%r12,%r15,8)	22,..
			0x400f21	60	vmovupdy %ymm3, 0x40(%r12,%r15,8)	112..
			0x400f28	60	vmovupdy %ymm4, 0x60(%r12,%r15,8)	13,..
			0x400f2f	60	vmovupdy %ymm5, 0x80(%r12,%r15,8)	103..
			0x400f39	60	vmovupdy %ymm6, 0xa0(%r12,%r15,8)	20,..
			0x400f43	60	vmovupdy %ymm7, 0xc0(%r12,%r15,8)	139..
			0x400f4d	60	vmovupdy %ymm8, 0xe0(%r12,%r15,8)	20,..
			0x400f57	59	add \$0x20, %r15	98,..
			0x400f5b	59	add \$0x100, %r14	15,..

71 double start=0.0, stop=0.0;

Nesse caso, houve vetorização. As operações usam instruções AVX:
vfmadd213pdy
vmovupdy



LAB02: Sten2D

- Compilar com `-O2 -qopt-report`
- Observar o relatório de otimização.

```
LOOP BEGIN at sten2d9pt_base.c(53,3)
```

```
remark #25096: Loop Interchange not done due to: Imperfect Loop Nest
```

```
remark #25452: Original Order found to be proper, but by a close margin
```

```
remark #15344: loop was not vectorized: vector dependence prevents vectorization
```

```
remark #15346: vector dependence: assumed OUTPUT dependence between fout[c] (68:9) and fout[c] (68:9)
```

```
remark #15346: vector dependence: assumed OUTPUT dependence between fout[c] (68:9) and fout[c] (68:9)
```

```
LOOP BEGIN at sten2d9pt_base.c(54,5)
```

```
remark #15344: loop was not vectorized: vector dependence prevents vectorization
```

```
remark #15346: vector dependence: assumed FLOW dependence between fout[c] (68:9) and fin[c] (68:9)
```

```
remark #15346: vector dependence: assumed ANTI dependence between fin[c] (68:9) and fout[c] (68:9)
```

```
LOOP BEGIN at sten2d9pt_base.c(67,7)
```

```
remark #15344: loop was not vectorized: vector dependence prevents vectorization
```

```
remark #15346: vector dependence: assumed FLOW dependence between fout[c] (68:9) and fin[c] (68:9)
```

```
remark #15346: vector dependence: assumed ANTI dependence between fin[c] (68:9) and fout[c] (68:9)
```



LAB02: Sten2D

- Verificar se há dependências.
- Usar `#pragma ivdep` no loop mais interno
- Recompilar e observar o relatório de otimização.

```
LOOP BEGIN at sten2d9pt_vect.c(53,3)
```

```
remark #25096: Loop Interchange not done due to: Imperfect Loop Nest
```

```
remark #25452: Original Order found to be proper, but by a close margin
```

```
remark #15542: loop was not vectorized: inner loop was already vectorized
```

```
LOOP BEGIN at sten2d9pt_vect.c(54,5)
```

```
remark #15542: loop was not vectorized: inner loop was already vectorized
```

```
LOOP BEGIN at sten2d9pt_vect.c(67,7)
```

```
<Peeled loop for vectorization>
```

```
remark #25015: Estimate of max trip count of loop=3
```

```
LOOP END
```

```
LOOP BEGIN at sten2d9pt_vect.c(67,7)
```

```
remark #15388: vectorization support: reference fout[c] has aligned access [ sten2d9pt_vect.c(68,9) ]
```

```
remark #15389: vectorization support: reference fin[c] has unaligned access [ sten2d9pt_vect.c(76,26) ]
```

```
remark #15389: vectorization support: reference fin[n] has unaligned access [ sten2d9pt_vect.c(74,26) ]
```



LAB02: Sten2D

- Alinhar os dados: `__mm_malloc/_mm_free`
- Usar `#pragma vector aligned` no loop mais interno
- Recompilar e observar o relatório de otimização.

OOP BEGIN at sten2d9pt_pad.c(83,7)

```
remark #15388: vectorization support: reference fout[c] has aligned access [ sten2d9pt_pad.c(84,9) ]
remark #15388: vectorization support: reference fin[c] has aligned access [ sten2d9pt_pad.c(92,26) ]
remark #15388: vectorization support: reference fin[n] has aligned access [ sten2d9pt_pad.c(90,26) ]
remark #15388: vectorization support: reference fin[s] has aligned access [ sten2d9pt_pad.c(91,26) ]
remark #15388: vectorization support: reference fin[e] has aligned access [ sten2d9pt_pad.c(89,26) ]
remark #15388: vectorization support: reference fin[w] has aligned access [ sten2d9pt_pad.c(88,26) ]
remark #15388: vectorization support: reference fin[nw] has aligned access [ sten2d9pt_pad.c(84,26) ]
remark #15388: vectorization support: reference fin[ne] has aligned access [ sten2d9pt_pad.c(85,26) ]
remark #15388: vectorization support: reference fin[sw] has aligned access [ sten2d9pt_pad.c(86,26) ]
remark #15388: vectorization support: reference fin[se] has aligned access [ sten2d9pt_pad.c(87,26) ]
```



NACAD

Núcleo Avançado de Computação de Alto Desempenho



AMT
High Performance Cloud Computing



IDENTIFICANDO OPORTUNIDADES DE DE VETORIZAÇÃO E PARALELISMO (MULTI-THREADING)



Identificando oportunidades de vetorização e paralelismo

Intel® Advisor fornece duas ferramentas que ajudam garantir que aplicações C/C++ e Fortran alcancem o potencial pleno de desempenho em processadores modernos, como processadores Intel® Xeon Phi™



Vectorization Advisor é uma ferramenta de otimização que permite identificar loops que mais beneficiariam da vetorização, identificar o que está bloqueando a vetorização, explorar o benefício das alternativas de reorganizações de dados, e verificar a corretude do código.

Threading Advisor é uma ferramenta de prototipagem Multi-threading que permite analisar, projetar, ajustar e verificar opções de paralelismo sem interromper o seu desenvolvimento normal.

Vectorization Advisor

Filtra quais loops foram vetorizados

Trip Counts

O que prejudica a vetorização?

Where should I vectorize and/or threading parallelism? Intel Advisor XE 2016

Summary Survey Refinement Reports Annotation Report Suitability Report

Elapsed time: 54.44s Vectorized Not Vectorized FILTER: All Modules Sources

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
							Vectorized	Efficiency	Vector Length
[loop at stl_algo.h:4740 in std:tr...]		0.170s	0.170s		Scalar	non-vectorizable loop ins...			
[loop at loopstl.cpp:2449 in s234_]	2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop at loopstl.cpp:2449 in s...]		0.150s	0.150s	12	Vectorized (Body)		AVX		4
[loop at loopstl.cpp:2449 in s...]		0.020s	0.020s	4	Remainder				
[loop at loopstl.cpp:7900 in vas_]		0.170s	0.170s	500	Scalar	vectorization possible but...			4
[loop at loopstl.cpp:3509 in s2...]	1 High vector register ...	0.160s	0.160s	12	Expand	Expand	AVX	~69%	8
[loop at loopstl.cpp:3891 in s279_]	2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%	8
[loop at loopstl.cpp:6249 in s414_]		0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
[loop at stl_numeric.h:247 in std...]	1 Assumed dependency...	0.150s	0.150s	49	Scalar	vector dependence preve...			

Foco nos laços que demoram mais CPU

Quais problemas de vetorização?

Quais instruções vetoriais estão sendo usadas?

Como eficiente é o código?



Buscando Vetorização Eficiente

1. Diagnóstico do compilador + Dados de Desempenho + Informações SIMD

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization	
			Loop Type	Why No Vectorization?
[loop in runCforallLambdaLoops]	0.094s	0.094s	Scalar	vector dependence prevents vector...
[loop in runCforallLambdaLoops]	0.140s	3.744s	Scalar	inner loop was already vectorized
[loop in std::Complex_base<double,struct _C_double_complex>::i...]	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop stats were reordered				
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...]	0.000s	0.234s	Scalar	nonstandard loop is not a vectoriza...

2. Detecta problemas e fornece recomendações de como resolve-los.

Issue: Peeled/Remainder loop(s) present

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at [Vector Essentials, Utilizing Full Vectors...](#)

Recommendation: Align memory access
Projected maximum performance gain: High
Projection confidence: Medium

The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

// Somewhere else
__assume_aligned(array, 32);
// Use array in loop
```

3. Análise de Dependência

Problems and Messages

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2	Read after write dependency	site2	dqtest2.cpp	dqtest2	✗ New
P3	Read after write dependency	site2	dqtest2.cpp	dqtest2	✗ New
P4	Write after write dependency	site2	dqtest2.cpp	dqtest2	✗ New
P5	Write after write dependency	site2	dqtest2.cpp	dqtest2	✗ New
P6	Write after read dependency	site2	dqtest2.cpp	dqtest2	✗ New
P7	Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	✗ New

4. Análise do padrão de acesso a memória

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cox:1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cox:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cox:925	No information available	100% / 0% / 0%	All unit strides

ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCRawLoops.cox:637	lcal.exe	
P23	0; 0	Unit stride	runCRawLoops.cox:638	lcal.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cox:628	lcal.exe	

P1: Onde a vetorização será mais benéfica.

Where should I add vectorization and/or threading parallelism?

Summary Survey Report Refinement Reports Annotation Report Suitability Report

time: 8,52s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources

Call Sites and Loops	🔥	💡 Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?
op at fractal.cpp:179 in <lambda1>::op...		💡 4 High vector ...	0,013s	12,020s	Collapse	Collapse
oop at fractal.cpp:179 in <lambda1>::o...	✓	💡 4 Serialized use...	0,013s	11,281s	Vectorized (Body)	
oop at fractal.cpp:179 in <lambda1>::o...	✓	💡 2 Data type co...	0,000s	0,163s	Peeled	
oop at fractal.cpp:179 in <lambda1>::o...	✓	💡 2 Data type co...	0,000s	0,576s	Remainder	
op at fractal.cpp:177 in <lambda1>::oper...	☐	💡 2 Data type co...	0,010s	12,030s	Scalar	

File: fractal.cpp:164 <lambda1>::operator()

Line	Source	Total Time	%	Loop Time
163	<pre> for (int x = x0; x < x1; ++x) { [loop at fractal.cpp:163 in <lambda1>::operator()] Scalar Loop. Not vectorized: outer loop was not auto-vectorized: consider us No loop transformations were applied </pre>			10.822s
164	<pre> for (int y = y0; y < y1; ++y) { [loop at fractal.cpp:164 in <lambda1>::operator()] Scalar Loop. Not vectorized: vectorization possible but seems inefficient. Us Loop was unrolled by 2 </pre>			10.822s
165	<pre> fractal_data_array[x - x0][y - y0] = calc_one_pixel(x, y, t </pre>	10.822s		
166	}			
167	}			
168	for (int y = y0, y_temp = 0; y < y1; ++y, ++y_temp) {			
169	area.set_pos(0, y - y0);			
170	for (int x = x0, x_temp = 0; x < x1; ++x, ++x_temp) {			
171	area.put_pixel(fractal_data_array[x_temp][y_temp]);			
172	}			



P2: Detecta problemas e fornece recomendações

Where should I add vectorization and/or threading parallelism? Intel Advisor XE 2016

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 8,81s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops		
						Vecto...	Estim...	Vector Len
[loop at market...]			11,460s	Scalar				
[loop at arena.cpp:88 in tbb::tbb::]		0,000s	11,460s	Scalar				
[loop at fractal.cpp:179 in <lambda1>::op ...]	5 Ineffective ...	0,000s	2,022s	Collapse	Collapse			
[loop at fractal.cpp:179 in <lambda1>::o ...]	2 Data type co ...	0,000s	2,022s	Remainder				

Click to see recommendation

Top Down Source Loop Assembly Assistance Recommendations Compiler Diagnostic Details

3 Issue: Ineffective peeled/remainder loop(s) present

5 All or some [source loop](#) iterations are not executing in the [loop body](#). Improve performance by moving source loop iterations from [peeled/remainder](#) loops to the loop body.

Disable unrolling

The [trip count](#) after loop unrolling is too small compared to [factor](#) using a [directive](#).

ICL/ICC/ICPC Directive	IFORT Directive
#pragma nounroll	!DIR\$ NOUNROLL
#pragma unroll	!DIR\$ UNROLL

Read More:

- [User and Reference Guide for the Intel C++ Compiler 15.0](#) > [Compiler Reference](#) > [Pragmas](#) > [Intel-specific Pragma Reference](#) > [unroll/nounroll](#).

Advisor shows hints to move iterations to vector body.



P3: Verifica dependência de dados

Check for loop-carried dependencies in your application

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_6	main	main.cpp:13	RAW:1 WAR:1 WAW:1	91% / 0% / 9%	Mixed strides

Detected dependencies

Memory Access Patterns Report Correctness Report

Problems and Messages

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	loop_site_6	main.cpp	test_1.exe	✓ Not a problem
P3	Read after write dependency	loop_site_6	crtexe.c; main.cpp	test_1.exe	✗ New
P4	Write after write dependency	loop_site_6	crtexe.c; main.cpp	test_1.exe	✗ New
P5	Write after read dependency	loop_site_6	crtexe.c; main.cpp	test_1.exe	✗ New

Write after read dependency: Code Locations

ID	Description	Source	Function	Module	State
X17	Read	main.cpp:22	main	test_1.exe	✗ New
<pre> 20 k += a[9]; 21 k *= a[8]; 22 k -= a[7]; 23 k += a[6]; 24 k *= a[5]; </pre>					
X18	Read	main.cpp:23	main	test_1.exe	✗ New
<pre> 21 k *= a[8]; 22 k -= a[7]; 23 k += a[6]; </pre>					

Source lines with Read and Write accesses detected



P4: Análise do padrão de acesso a memória

Stride distribution

Check memory access patterns in your application Intel Advisor XE 2016

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_79	operator()	fractal.cpp:179	No information available	100% / < 1,0000% / ...	Mixed strides
loop_site_93	operator()	fractal.cpp:179	No information available	100% / 0% / 0%	All unit strides
loop_site_94	operator()	fractal.cpp:179	No information available	100% / 0% / 0%	All unit strides

All memory accesses are uniform, with zero unit stride, so the same data is read in each iteration

We can therefore declare this function using the omp syntax: `pragma omp declare simd uniform(x0`

Memory Access Patterns Report

ID	Stride	Type	File	Line
P18	0	Unit stride	fractal.cpp:66	fractal.exe
P21	0	Unit stride	fractal.cpp:66	fractal.exe
P24	0	Unit stride	fractal.cpp:68	fractal.exe
P27	0	Unit stride	fractal.cpp:69	fractal.exe
P30	0	Unit stride	fractal.cpp:74	fractal.exe

```

64     color_t color;
65
66     fx0 = x0 - size_x / 2.0f;
67     fy0 = y0 - size_y / 2.0f;
68     fx0 = fx0 / magn + cx;
69     fy0 = fy0 / magn + cy;
70

```



LAB 03: Vectorization Advisor

- **Objetivos:**
 - Identificar loops que mais serão beneficiados pela vetorização.
 - Identificar o que está bloqueando vetorização.
 - Aumentar a confiança de que a vetorização é segura.
 - Explorar o benefício da reorganizações de dados.
- **Arquivos:**
 - `handson/lab03/pricing.tar.gz`
 - Copie para sua area de trabalho e segue as instruções dada.
- **Tutorial detalhando disponível em:**
 - <https://software.intel.com/en-us/intel-advisor-tutorial-vectorization-linux-cplusplus>



LAB 03

Código de exemplo - Black-Scholes Pricing Code

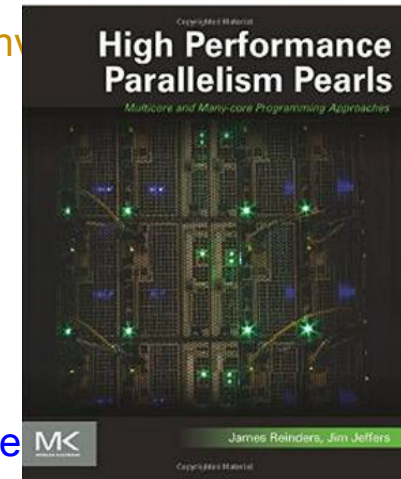
“a mathematical model of a financial market containing certain derivative instruments.”

Exemplo retirado do livro “High Performance Parallelism Pearls”

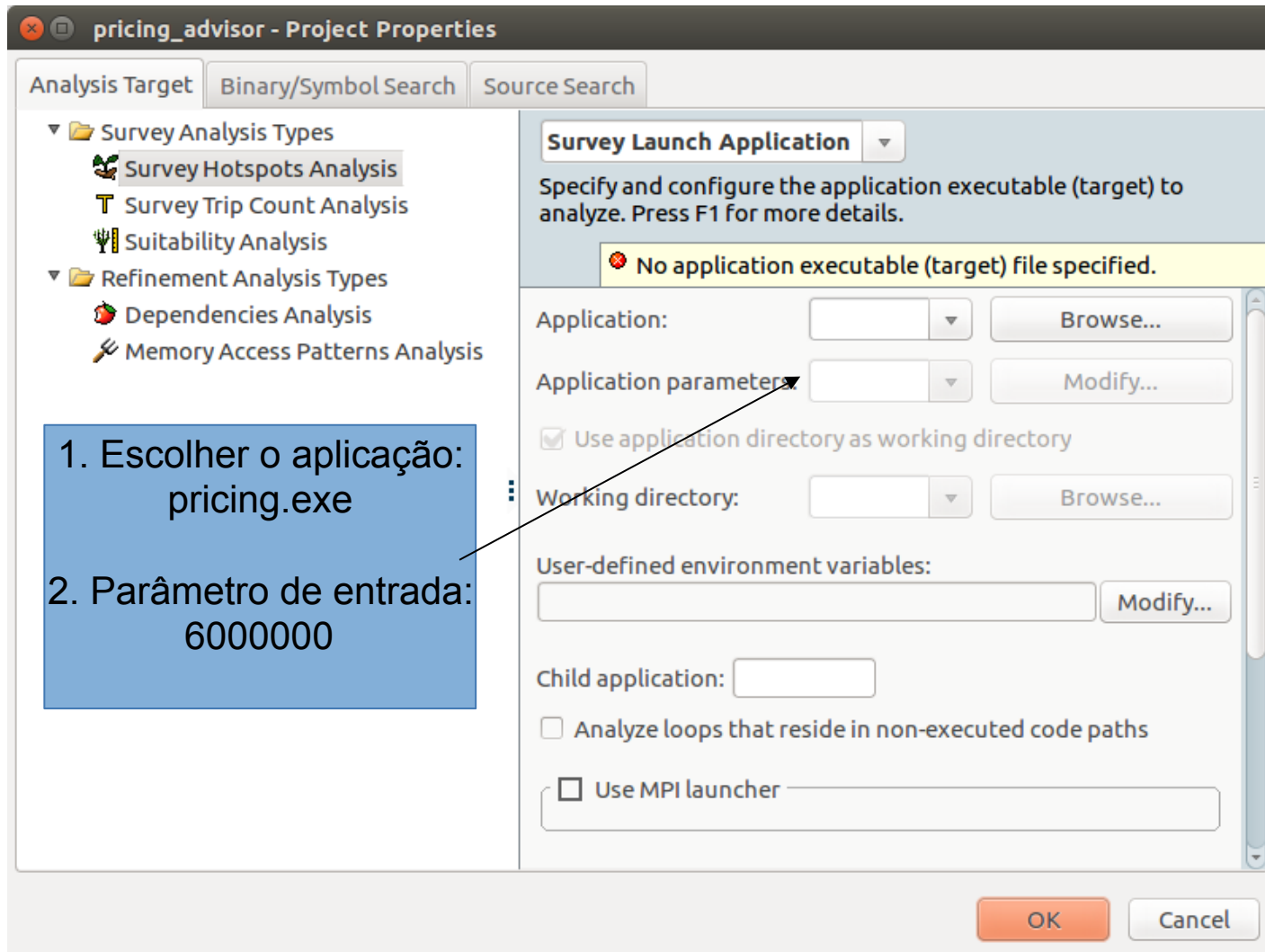
Código fonte: <http://lotsofcores.com/pearls.code>

Artigo sobre otimização deste método

<https://software.intel.com/en-us/articles/case-study-computing-black-scholes-advanced-vector-extensions>



Criando Projeto



pricing_advisor - Project Properties

Analysis Target | Binary/Symbol Search | Source Search

- Survey Analysis Types
 - Survey Hotspots Analysis
 - Survey Trip Count Analysis
 - Suitability Analysis
- Refinement Analysis Types
 - Dependencies Analysis
 - Memory Access Patterns Analysis

Survey Launch Application

Specify and configure the application executable (target) to analyze. Press F1 for more details.

No application executable (target) file specified.

Application: Browse...

Application parameters: Modify...

Use application directory as working directory

Working directory: Browse...

User-defined environment variables: Modify...

Child application:

Analyze loops that reside in non-executed code paths

Use MPI launcher

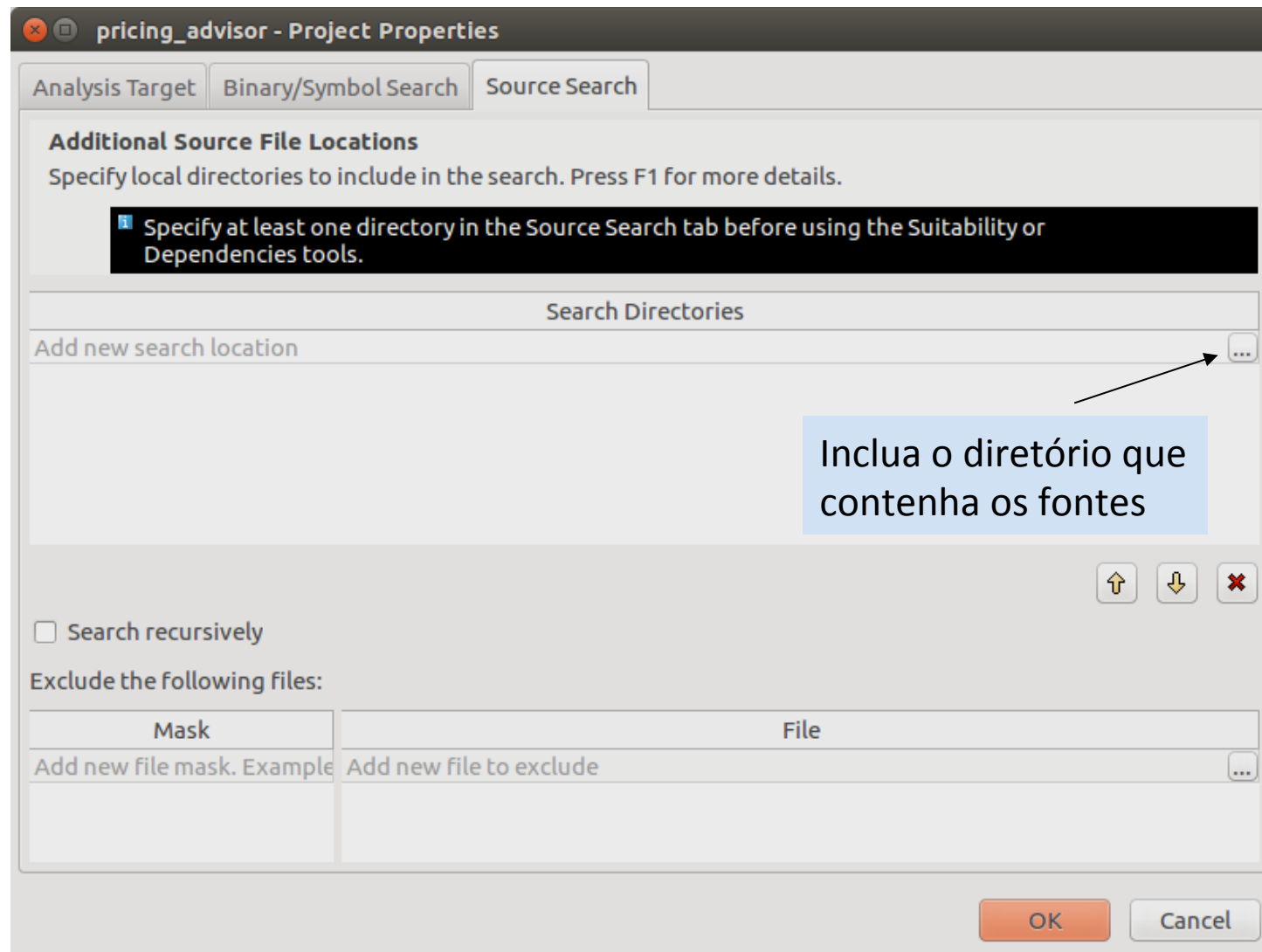
OK Cancel

1. Escolher o aplicação:
pricing.exe

2. Parâmetro de entrada:
6000000



Indicar o diretório do código-fonte



Survey Analysis

Intel Advisor 2017 interface showing the Survey Report section. The interface includes a sidebar with workflow steps, a top navigation bar with filters, and a main content area with various metrics and a table of loops. Three blue circles with numbers 1, 2, and 3 are overlaid on the interface, pointing to the 'Collect' button, the 'Loop metrics' bar chart, and the 'Survey Report' tab respectively.

1. Survey Target

1.1 Find Trip Counts and...

Mark Loops for Deeper A...

Select loops in the Survey Report for Dependencies and/or Memory Access Patterns analysis.

-- There are no marked loo...

2.1 Check Dependencies

-- Nothing to analyze --

2.2 Check Memory Acce...

-- Nothing to analyze --

Elapsed time: 3,30s Vectorized Not Vectorized OFF Smart Mode

FILTER: All Modules All Sources Loops All Threads

Summary Survey Report Refinement Reports

Vectorization Advisor

Vectorization Advisor is a vectorization analysis tool that lets you identify loops that will benefit most from vectorization.

Program metrics

Elapsed Time: 3,30s
Vector Instruction Set: None
Number of CPU Threads: 1

Loop metrics

Total CPU time	3,29s	100,0%
Time in 0 vectorized loops	0s	
Time in scalar code	3,29s	100,0%

Vectorization Gain/Efficiency (Not available)

Top time-consuming loops

Loop	Source Location	Self Time	Total Time
GetOptionPrices	pricing1.cpp:46	0,600s	3,280s
main	pricing1.cpp:74	0,010s	0,010s

Collection details

Platform information

CPU Name: Intel(R) Core(TM) i5-4210U CPU @ 1.70GHZ
Frequency: 2,40 GHz
Logical CPU Count: 4

Passos:

1. Coletar os dados
2. Visualizar o resumo
3. Analisar o relatório de sustentabilidade

Survey Report

Elapsed time: 3,42s Vectorized Not Vectorized OFF Smart Mode

FILTER: All Modules | All Sources | Loops | All Threads

Summary | **Survey Report** | Refinement Reports

Higher instruction set architecture (ISA) available
Consider recompiling your application using a higher ISA.

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vector
[loop in GetOptionPrices at pricing1.c...	2 Assumed depen...	0,496s	3,388s	Scalar	vector dependence prevents ...	
[loop in main at pricing1.cpp:74]	1 Assumed depen...	0,022s	0,022s	Scalar	vector dependence prevents ...	

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

File: pricing1.cpp:46 GetOptionPrices

Line	Source	Total Time	%	Loop Time	%	Traits
44	float d1, d2, p1, p2;					
45						
46	for (i = 0; i < N; i++)			3,388s		
47	{					
48	d1 = (log(pS0[i] / pK[i]) + (r + sig * sig * 0.5) * pT[i]) / (sig * ...	0,464s				
49	d2 = (log(pS0[i] / pK[i]) + (r - sig * sig * 0.5) * pT[i]) / (sig * ...					
50	p1 = cdfnormf(d1);	1,306s				
51	p2 = cdfnormf(d2);	1,106s				
52	pC[i] = pS0[i] * p1 - pK[i] * exp((-1.0) * r * pT[i]) * p2;	0,512s				
53	}					
Selected (Total Time):		0s				

Recomendações

The screenshot displays the Intel Advisor 2017 interface. The top toolbar includes options for 'Elapsed time: 3,42s', 'Vectorized', 'Not Vectorized', and 'Smart Mode'. The left sidebar shows the 'Vectorization Workflow' with steps: 1. Survey Target, 1.1 Find Trip Counts and..., Mark Loops for Deeper A..., 2.1 Check Dependencies, and 2.2 Check Memory Accesses. The main area shows a table of vectorization issues and a detailed view of two recommendations.

INTEL ADVISOR 2017

Elasped time: 3,42s Vectorized Not Vectorized Smart Mode

FILTER: All Modules | All Sources | Loops | All Threads

Summary | Survey Report | Refinement Reports

Higher instruction set architecture (ISA) available
Consider recompiling your application using a higher ISA.

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vector
[loop in GetOptionPrices at pricing1.c...	2 Assumed dependen...	0,496s	3,388s	Scalar	vector dependence prevents ...	
[loop in main at pricing1.cpp:74]	1 Assumed dependen...	0,022s	0,022s	Scalar	vector dependence prevents ...	

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

All recommendations: C++, FORTRAN

Issue: Assumed dependency present
The compiler assumed there is an anti-dependency (Write after read - WAR) or true dependency (Read after write - RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Recommendation: Confirm dependency is real Confidence: Need More Data
There is no confirmation that a real (proven) dependency is present in the loop. To confirm: Run a [Dependencies analysis](#).

Issue: Data type conversions present
There are multiple data types within loops. Utilize hardware vectorization support more effectively by avoiding data type conversion.

Recommendation: Use the smallest data type Confidence: Low
The [source loop](#) contains data types of different widths. To fix: Use the smallest data type that gives the needed precision to use the entire [vector register width](#).
Example: If only 16-bits are needed, using a short rather than an int can make the difference between eight-way or four-way SIMD parallelism, respectively.

Checar dependências

The screenshot displays the Intel Advisor 2017 interface. The top toolbar includes icons for various analysis tasks. The main window shows a summary of a loop analysis. The 'Batch mode' is turned off. The 'Vectorization Workflow' and 'Threading Workflow' are visible on the left sidebar. The main content area shows a table with the following data:

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Max. Site Footprint
[loop in GetOptionPrices at pricing3.cpp:...	No dependencies found	No information available	No information available	No information available

Below this table, the 'Problems and Messages' section is expanded, showing a single problem:

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	loop_site_4	pricing3.cpp	pricing.exe	✓ Not a problem

The 'Parallel site information: Code Locations' section is also expanded, showing the following code snippet:

```

X1 0x401187 Parallel site
45 sqrt2 = sqrtf(2.0f);
46
47 for (i = 0; i < N; i++)
48 {
  
```

The right sidebar shows a 'Filter' section with the following items:

- Severity: Information (1 item)
- Type: Parallel site infor... (1 item)
- Source: pricing3.cpp (1 item)
- Module: pricing.exe (1 item)
- State: Not a problem (1 item)

At the bottom right, there is a 'Sort By Item Name' button and a close icon.

Implementar as recomendações

The screenshot shows the Intel Advisor 2017 interface. The top bar indicates 'Elapsed time: 3,42s' and 'Vectorized' status. The left sidebar contains navigation options like 'Vectorization Workflow', 'Threading Workflow', and 'Batch mode'. The main area displays a warning: 'Higher instruction set architecture (ISA) available. Consider recompiling your application using a higher ISA.' Below this is a table of vectorization issues.

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?
[loop in GetOptionPrices at pricing1.c...	2 Assumed depen ...	0,496s	3,388s	Scalar	vector dependence prevents ...
[loop in main at pricing1.cpp:74]	1 Assumed depen ...	0,022s	0,022s	Scalar	vector dependence prevents ...

The 'Why No Vectorization?' column for the second loop is expanded, showing the 'Recommendations' tab. The text reads: 'Run a Dependencies analysis to check if the loop has real dependencies. There are two types of dependencies: True dependency - Read after write (RAW), Anti-dependency - Write after read (WAR). If no dependencies exist, use one of the following to tell the compiler it is safe to vectorize: Directive to prevent all dependencies in the loop, Directive to ignore only vector dependencies (which is safer), restrict keyword. If anti-dependency exists, use a directive where k is smaller than the distance between dependent items in anti-dependency. This enables vectorization, as dependent items are put into different vectors.'

Target	ICL/ICC/ICPC Directive	IFORT Directive
Source Loop	#pragma simd or #pragma omp simd	IDIR\$ SIMD or !\$OMP SIMD

←

Target	ICL/ICC/ICPC Directive	IFORT Directive
Source Loop	#pragma ivdep	IDIR\$ IVDEP

Target	ICL/ICC/ICPC Directive	IFORT Directive
Source Loop	#pragma simd vectorlength(k)	IDIR\$ SIMD VECTORLENGTH(k)

Rodar Survey Analysis

Elapsed time: 3,14s **Vectorized** **Not Vectorized** **OFF** Smart Mode² **INTEL ADVISOR 2017**

FILTER: All Modules | All Sources | Loops | All Threads

Summary | Survey Report | Survey Source: pricing2.cpp | Refinement Reports

Source | Assembly | Stack

File: pricing2.cpp:46 GetOptionPrices

Line	Source	Total Time	%	Loop Time	%	Traits
42	{					
43	int i;					
44	float d1, d2, p1, p2;					
45				3,120s		
46	for (i = 0; i < N; i++)					
47	{					
48	// conversao de double para float					
49	d1 = (logf(pS0[i] / pK[i]) + (r + sig * sig * 0.5f) * pT[i]) /	0,472s				
50	d2 = (logf(pS0[i] / pK[i]) + (r - sig * sig * 0.5f) * pT[i]) /					
51	p1 = cdfnormf(d1);	1,044s				
52	p2 = cdfnormf(d2);	1,112s				
53	pC[i] = pS0[i] * p1 - pK[i] * expf((-1.0f) * r * pT[i]) * p2;	0,492s				
54	}					
55	}					
56						
57	double start, finish;					
58	double t;					
59						
60	int main(int argc, char *argv[])					
61	{					
		Selected (Total Time):	1,044s			

Annotations Example

$$\text{cdfnormf}(x) = 0.5f + 0.5f * \text{erf}(x)$$

Uso de Instruções Vetoriais AVX

The screenshot displays the Intel Advisor 2017 interface. The top toolbar includes options for 'Elapsed time: 0,14s', 'Vectorized', 'Not Vectorized', 'OFF', and 'Smart Mode'. The left sidebar shows the 'Vectorization Workflow' with sections for '1. Survey Target', '1.1 Find Trip Counts and...', 'Mark Loops for Deeper A...', '2.1 Check Dependencies', and '2.2 Check Memory Acce...'. The main area shows a table of vectorization results:

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?
[loop in main at pricing5.cpp:83]		0,010s	0,010s	Vectorized (Body)	SSE
[loop in GetOptionPrices at pricing5.cpp:49]	1 Potential u...	0,008s	0,120s	Vectorized (Bo...	SSE


Below the table, a detailed issue is shown: 'Issue: Potential underutilization of FMA instructions'. The text explains that the current hardware supports the AVX2 instruction set architecture (ISA), which enables the use of fused multiply-add (FMA) instructions. A recommendation is provided to target the AVX2 ISA, with a confidence level of 'Low'. The recommendation text states: 'Although static analysis presumes the loop may benefit from FMA instructions available with the AVX2 ISA, no AVX2-specific code executed for this loop. To fix: Use the `-xCORE-AVX2` compiler option to generate AVX2-specific code, or the `-axCORE-AVX2` compiler option to enable multiple, feature-specific, auto-dispatch code generation, including AVX2.'

Windows* OS	Linux* OS
/QxCORE-AVX2 or /QaxCORE-AVX2	-xCORE-AVX2 or -axCORE-AVX2

Read More:

- [ax, Qax; x, Qx](#)
- [Code Generation Options](#) in the [Intel® C++ Compiler 16.0 User and Reference Guide](#)
- [Compiling for the Intel® Xeon Phi™ processor x200 and the Intel® AVX-512 ISA](#) and [Vectorization Resources for Intel® Advisor Users](#)

Relatorio Final

Welcome e000 

Vectorization Workflow | **Threading Workflow**

Elapsed time: 0,09s | Vectorized | Not Vectorized | Smart Mode

FILTER: All Modules | All Sources | Loops | All Threads

Summary | Survey Report | Refinement Reports

Vectorization Advisor

Vectorization Advisor is a vectorization analysis tool that lets you identify loops that will benefit most from vectorization.

Program metrics
 Elapsed Time: 0,09s
 Vector Instruction Set: AVX
 Number of CPU Threads: 1



Loop metrics

Total CPU time	0,08s	<div style="width: 100%;"></div>	100,0%
Time in 2 vectorized loops	0,08s	<div style="width: 100%;"></div>	100,0%
Time in scalar code	0s		

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	8,43x	<div style="width: 100%;"></div>	~100%
Program Theoretical Gain	8,43x		

Top time-consuming loops

Loop	Source Location	Self Time ³⁾	Total Time ³⁾
 main	pricing5.cpp:83	0,012s	0,012s
 GetOptionPrices	pricing5.cpp:49	0,008s	0,068s

Collection details

Platform information
 CPU Name: Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz

1. Survey Target

1.1 Find Trip Counts and...

Mark Loops for Deeper A...

Select loops in the Survey Report for Dependencies and/or Memory Access Patterns analysis.

-- There are no marked loo...

2.1 Check Dependencies

-- Nothing to analyze --

2.2 Check Memory Ace...

-- Nothing to analyze --



Identificando Oportunidades de Paralelismo

Inserindo as “anotações” do Advisor para executar a próxima fase: **Check Suitability**

```

__declspec(noinline) void GetOptionPrices(float *pT, float *pK, float *pS0, float *pC)
{
    int i;
    float d1, d2, p1, p2;

    // #pragma cilk grainsize=256
    // #pragma omp parallel for
    #pragma ivdep
    #pragma unroll(4)
    ANNOTATE_SITE_BEGIN( MySite1 ); // Place before the loop control statement to begin a parallel code region (parallel site).
    for (i = 0; i < N; i++)
    {
        ANNOTATE_ITERATION_TASK( MyTask1 ); // Place at the start of loop body. This annotation identifies an entire body as a task.
        /* - Float usage */
        d1 = (logf(pS0[i] / pK[i]) + (r + sig * sig * 0.5f) * pT[i]) / (sig * sqrtf(pT[i]));
        d2 = (logf(pS0[i] / pK[i]) + (r - sig * sig * 0.5f) * pT[i]) / (sig * sqrtf(pT[i]));
        p1 = cdfnormf(d1);
        p2 = cdfnormf(d2);
        pC[i] = pS0[i] * p1 - pK[i] * exp((-1.0f) * r * pT[i]) * p2;
    }
    ANNOTATE_SITE_END(); // End the parallel code region, after task execution completes
}
}

```

Linux - Compilando / Link com Advisor
 icpc -O2 -openmp 02_pricing.cpp
 -o pricing.exe
 -I/opt/intel/advisor_xe/include/
 -L/opt/intel/advisor_xe/lib64/

Identificando Oportunidades de Paralelismo

Welcome e000 ✕

Elapsed time: 1,54s **Vectorized** Not Vectorized **OFF** Smart Mode⁹ **INTEL ADVISOR 2017**

FILTER: All Modules All Sources Loops All Threads

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Maximum Program Gain For All Sites: 1,34x

Target System: CPU Threading Model: Intel Cilk Plus CPU Count: 2

Site Label	Source Location	Impact to Program Gain	Combined Site Metrics, All Instances			Site Instance Metrics, Parallel Ti...
			Total Serial ...	Total Parallel ...	Site G...	Site Instance Metrics, Parallel Ti...
solve	pricing6.c...	1,34x	1,244s	0,916s	1,36x	0,916s

1. Survey Target

Collect

1.1 Find Trip Counts and...

Collect

2. Annotate Sources

Add Intel Advisor XE annotations to identify possible parallel tasks and their enclosing parallel sites.

Steps to annotate

3. Check Suitability

Collect

4. Check Dependencies

Collect

-- Nothing to analyze --

Site Performance Scalability **Site Details**

Scalability of Maximum Site Gain

CPU Count	Maximum Site Gain
2	1x
4	2x
8	4x
16	8x
32	16x
64	32x

Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks): 6000000

Avg. Iteration (Task) Duration: < 0,000s

0,008x
0,040x
0,200x
1x (6000000)
5x
25x
125x

0,008x
0,040x
0,200x
1x (< 0,000s)
5x
25x
125x

Runtime Modeling

Type of Change Gain Be

- Reduce [Site Overhead](#)
- Reduce [Task Overhead](#)
- Reduce [Lock Overhead](#)
- Reduce [Lock Contention](#)
- Enable [Task Chunking](#)

Apply

16.1% Load Imbalance: 0,147s

Min Task Time: < 0,000s
Max Task Time: < 0,000s

32.1% Runtime Overhead: 0,294s

Region (Site) Overhead: < 0.000s

Warning

⚠ Current tasks are too fine-grain, and not effective for multi-threading.



LAB 03: Threading Advisor

- Objetivos:
 - Identificar oportunidades de paralelismo multithread
- Arquivos
 - `handson/lab03/nqueens_advisor`
 - Copie para sua area de trabalho e segue as instruções dada.
- Tutorial detalhando está disponível em:
 - <https://software.intel.com/en-us/intel-advisor-tutorial-vectorization-linux-cplusplus>



Mais Informações

- Modernização de códigos:
 - <https://software.intel.com/en-us/modern-code>
- Books:
 - Intel Xeon Phi Processor High Performance programming: Knights Landing Edition
 - High Performance Parallelism Pearls
 - lotsofcores.com

