



# Escola Supercomputador SDUMONT

## MP11: Introdução a programação com OpenACC

Pedro Pais Lopes

01/08/2017

# Agenda

- Parte 1: teoria
- Parte 2: compiladores
- Parte 3: pré-prática
- Parte 4: prática



# Parte 1: TEORIA

- O acelerador
- Definição de OpenACC
- Motivação e portabilidades
- Modelo de memória e execução
- Diretivas
- Pequeno exemplo



# Ponto de largada!

OpenACC não é pra você

N M M Meu programa não tem laço!





# O acelerador

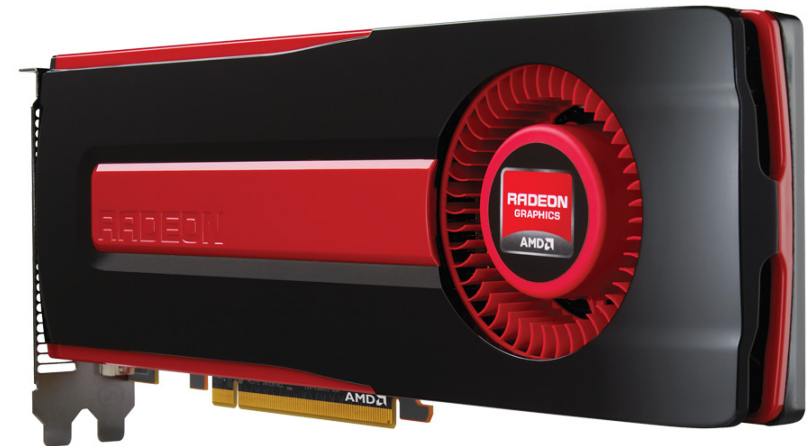
- Acelerador: dispositivo que realiza cálculos matemáticos massivos, muito específico e externo ao conjunto CPU-Memória
- Em geral a inequação abaixo é respeitada

$$n\text{Cores}_{\text{acelerador}} \ggg n\text{Cores}_{\text{CPU}}$$

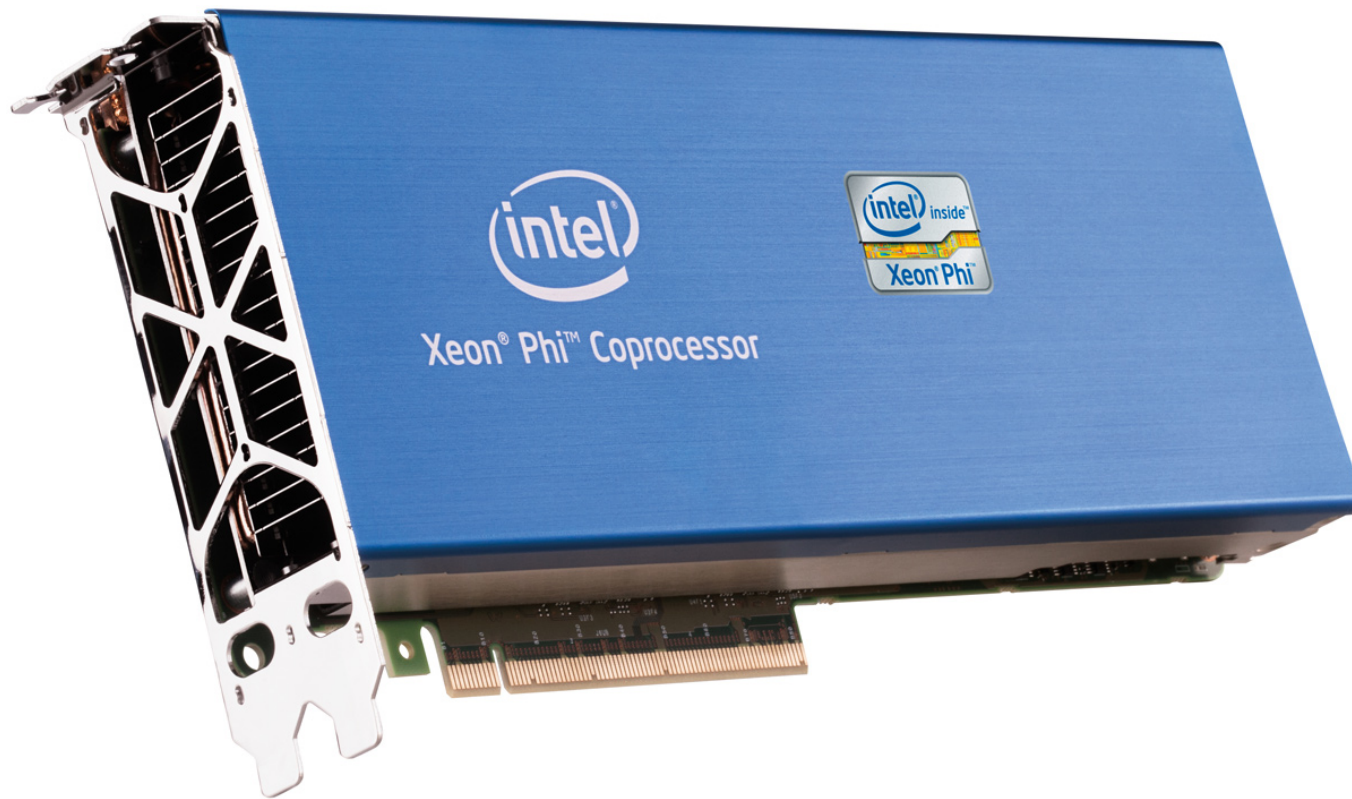


# Exemplo

- Mais famoso: GPGPU (General Purpose computation on Graphics Processing Units)

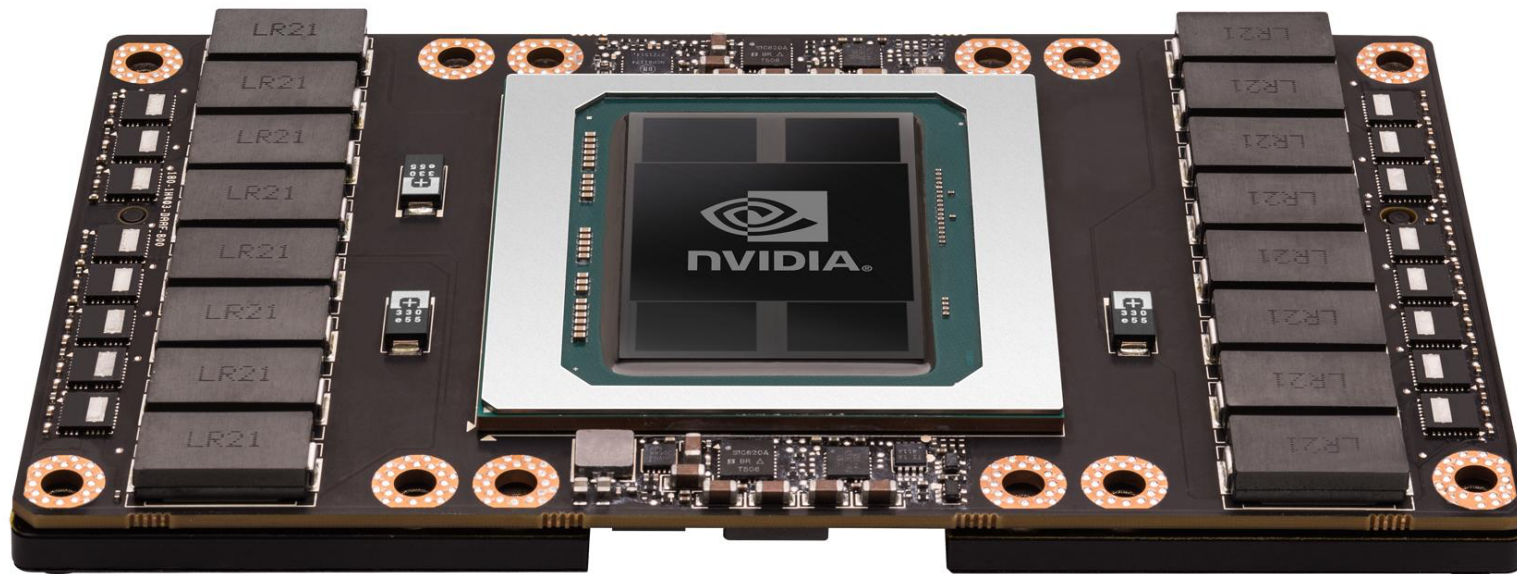


# Outro exemplo



EXAFLOP SISTEMAS

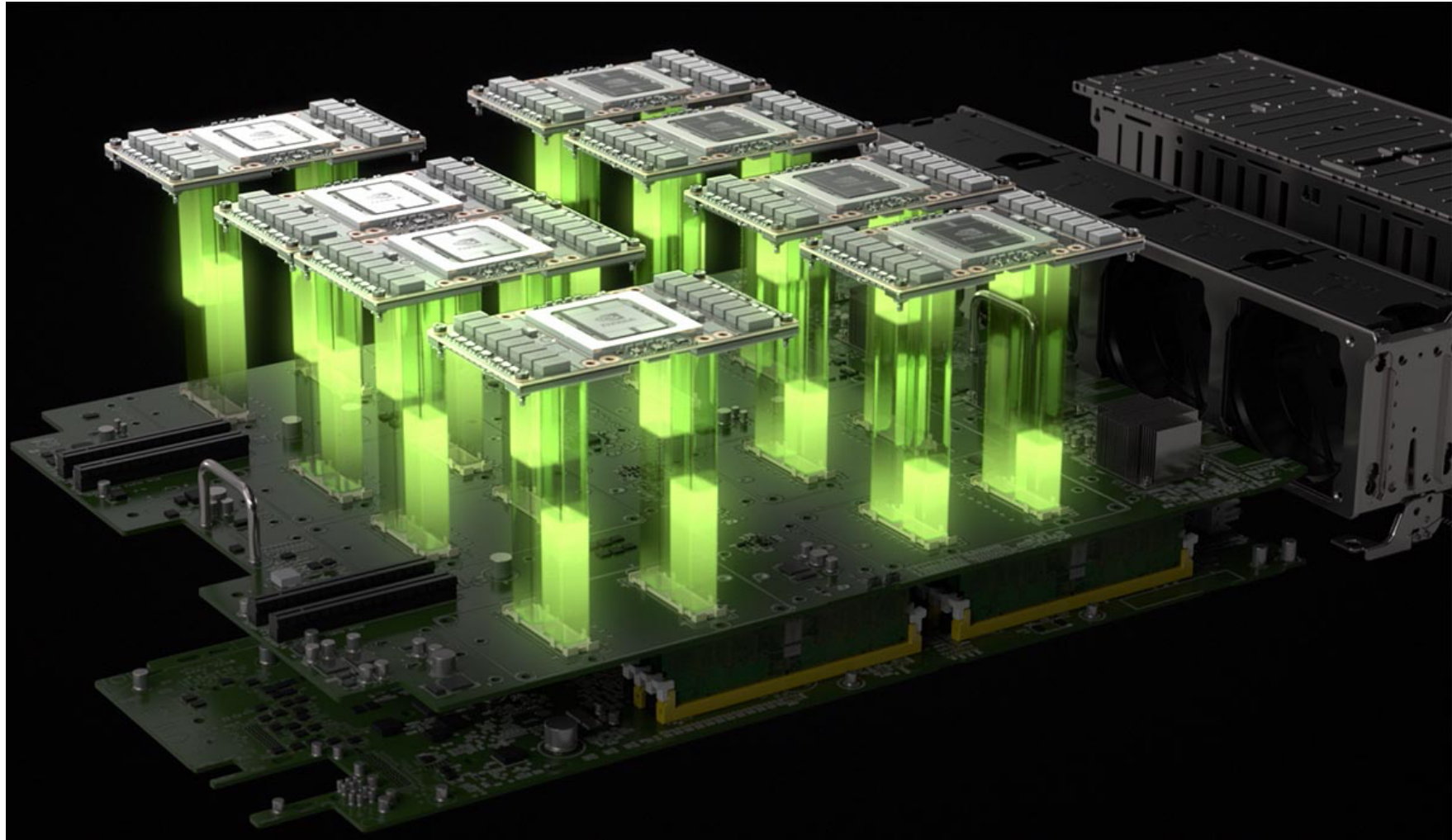
# Mais um (onde "espeto" isso?)



EXAFLOP SISTEMAS



# Aqui!



28.672 CORES = 122.400.000.000 transistores (GPU)  
40 CORES = 14.400.000.000 transistores (CPU)





US\$ 149k

40.960 CORES = 168.000.000.000 transistores (GPU)

44 CORES = 14.400.000.000 transistores (CPU)

# Quero acelerar meu código, o que escolho?

FACILIDADE

VELOCIDADE

BIBLIOTECAS

OpenACC

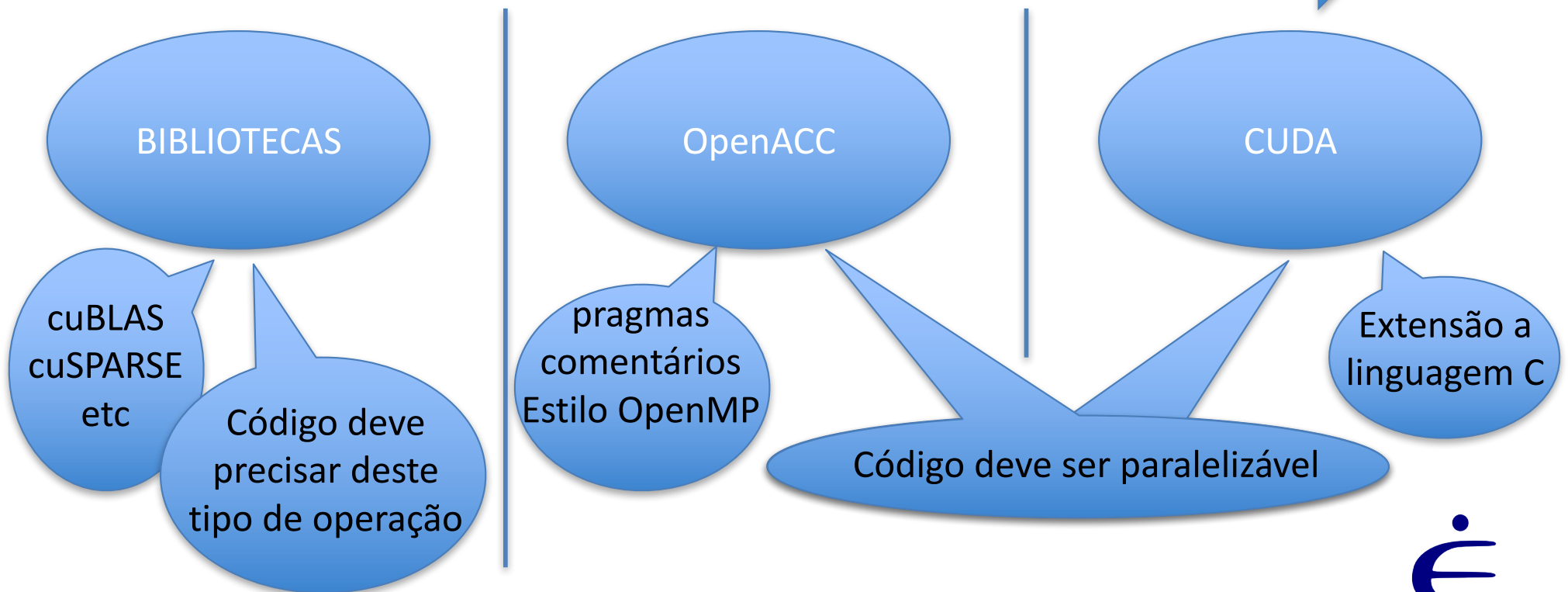
CUDA



EXAFLOP SISTEMAS

# Como levar este poder de processamento ABUNDANTE para sua aplicação?

DIFICULDADE  
INTRUSÃO AO CÓDIGO  
DESEMPENHO





# Directive-based Accelerator Programming With OpenACC (Mathew Colgrove, PGI, 2013)

## Matrix Multiply Source Code Size Comparison:

```
1 void
2 compute_m1_xcopy_from_A2(BM, float A2[BM], float B2[BM],
3 int M1, int M2, int M3)
4 {
5     // OpenACC code region
6     {
7         // OpenACC code region
8         for (int k = 0; k < M3; k++) {
9             for (int j = 0; j < M2; j++) {
10                 C[j][k] = 0.0;
11             }
12             for (int i = 0; i < M1; i++) {
13                 for (int l = 0; l < M2; l++) {
14                     C[i][k] += A2[i][l] * B2[l][j];
15                 }
16             }
17         }
18     }
19 }
```

Directives

```
1 void
2 __global__ void compute_m1_xcopy_from_A2(BM, float A2[BM], float B2[BM],
3 int M1, int M2, int M3)
4 {
5     // OpenACC code region
6     {
7         // OpenACC code region
8         for (int k = 0; k < M3; k++) {
9             for (int j = 0; j < M2; j++) {
10                 C[j][k] = 0.0;
11             }
12             for (int i = 0; i < M1; i++) {
13                 for (int l = 0; l < M2; l++) {
14                     C[i][k] += A2[i][l] * B2[l][j];
15                 }
16             }
17         }
18     }
19 }
```

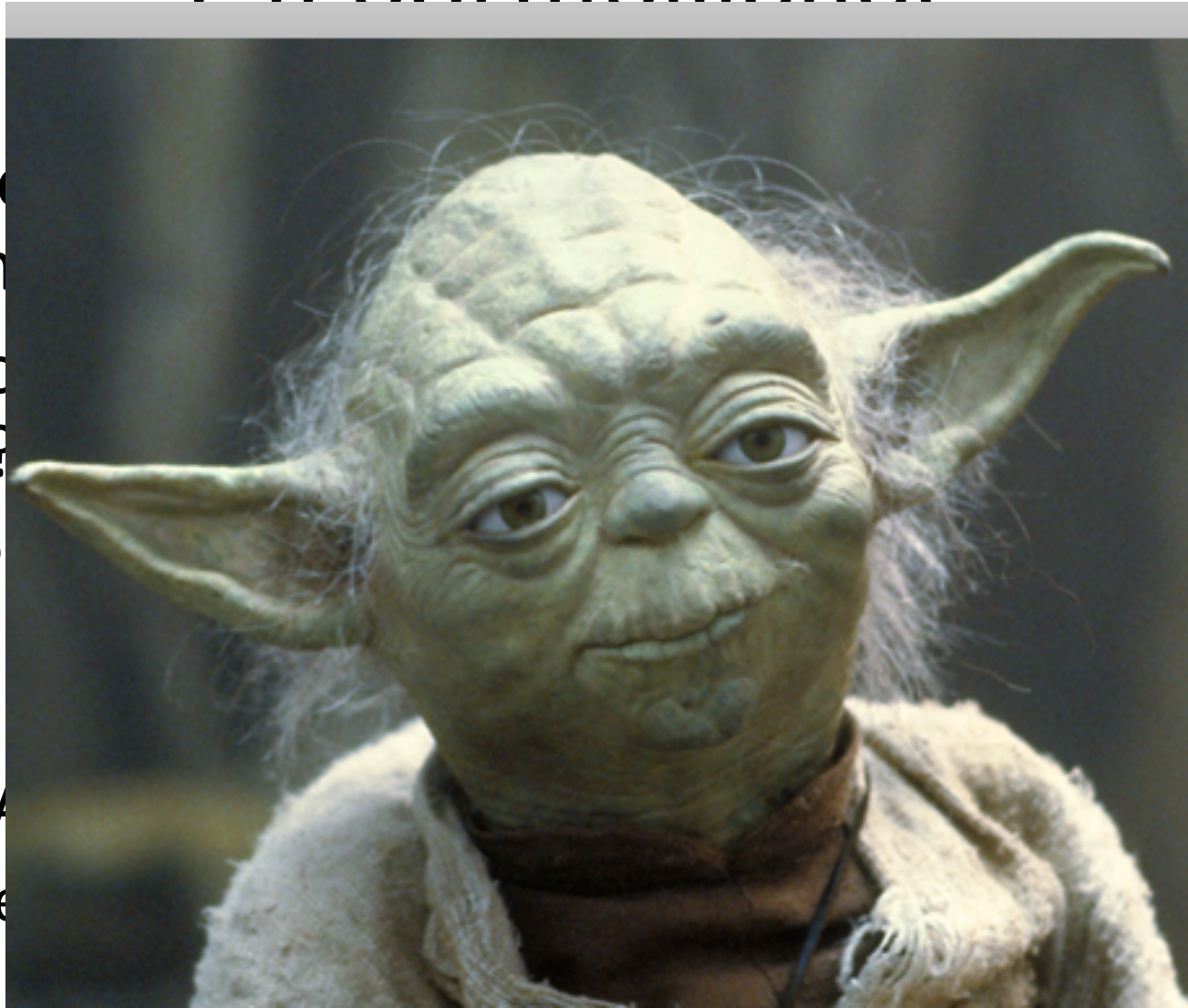
CUDA C

```
1 void compute_m1_xcopy_from_A2(BM, float A2[BM], float B2[BM],
2 int M1, int M2, int M3)
3 {
4     // OpenACC code region
5     {
6         // OpenACC code region
7         for (int k = 0; k < M3; k++) {
8             for (int j = 0; j < M2; j++) {
9                 C[j][k] = 0.0;
10             }
11             for (int i = 0; i < M1; i++) {
12                 for (int l = 0; l < M2; l++) {
13                     C[i][k] += A2[i][l] * B2[l][j];
14                 }
15             }
16         }
17     }
18 }
```

OpenCL

# E a portabilidade?

- Existe “com
- Um d CPU?
  - Ro
  -
- Tenh CUDA
  - “Se
  -



smo

o sem

CPU

em



EXAFLOP SISTEMAS



# E a portabilidade?

- Quais compiladores entendem código CUDA?
  - Temos o da NVIDIA (que usa intensamente o LLVM) e o da PGI (que entende Fortran mas acredito não entender C)
  - E as outras plataformas?
- Como fica a linguagem Fortran?
  - Temos o compilador da PGI, com o CUDA-Fortran



# Problema similar no passado

- Programação para máquinas de memória compartilhada
  - Conceito mais comum: threads
- Cada vendor tinha sua forma de programação para ambientes multiprocessados
  - Programa paralelo da SGI não roda na IBM, que não roda na Fujitsu, que não roda na Compaq, nem na HP-UX
- Em 1997: OpenMP v1.0 para Fortran, e em 1998 OpenMP para C
  - Resultado: padronização



# Padrão completo e simples

- Diretivas de compilação
  - Pragmas em C, comentários em Fortran
- Largamente aceito
  - GNU, IBM, Oracle, Intel, PGI, Absoft, Lahey/Fujitsu, PathScale, HP, Microsoft, Cray, OpenUH e vários outros
- Antigo, porém muito atual
  - Antes utilizado nos antigos IBM SP2 e SUN SPARC com centenas de CPUs e hoje utilizado nas CPUs multi-core!



# O OpenACC



- Fácil: diretivas simples, alto nível, muito próxima (da sintaxe) do OpenMP, suporta C, C++ e Fortran
- Aberta: não está atrelada a um *vendor* ou a um compilador
- Poderoso: abstração do paralelismo por parte do compilador permite otimização



# O OpenACC



- Diretivas para especificar regiões do código que devem ser paralelizadas e executadas em um acelerador
- Modelo básico: o *host* controla o processamento no *target*
  - *Host*: CPU, sua cache, sua memória e onde é executado um sistema operacional
  - *Target*: um acelerador
- Cria, portanto, programas heterogêneos de alto nível
  - Sem inicialização explícita
  - Sem transferências explícitas entre host e acelerador





# O OpenACC



- Interoperacionalidade e compatibilidade com linguagens específicas dos aceleradores (OpenCL, CUDA, COI\*)
- Região de memória no acelerador com resultados de trechos acelerados com OpenACC pode ser utilizados pelas linguagens e vice-versa
- Resultado: foco do programador em expor o paralelismo
  - Modificar laços para aproveitar os muitos núcleos
  - Aproveitar cache, memória compartilhada no “*stream multiprocessor*”, aumentar a “*spatial locality*” no acesso a memória
- Consequência comum: código no host também sofre melhoria no desempenho
- Se compilador não aceita OpenACC: ignora as diretivas e compila o código -> não limita a portabilidade original!

\* Coprocessor Offload Infrastructure

# Exemplo: calculando PI

```
#include <stdio.h>
#define N 1000000000
int main(void) {
    double pi = 0.0f; long i;

    for (i=0; i<N; i++)
    {
        double t=(double) ((i+0.5)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi=%f\n",pi/N);
    return 0;
}
```



# Exemplo: calculando PI com OpenMP

```
#include <stdio.h>
#define N 1000000000
int main(void) {
    double pi = 0.0f; long i;
    #pragma omp parallel for reduction(+: pi)

    for (i=0; i<N; i++)
    {
        double t=(double) ((i+0.5)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi=%f\n",pi/N);
    return 0;
}
```



# Exemplo: calculando PI com OpenACC

```
#include <stdio.h>
#define N 1000000000
int main(void) {
    double pi = 0.0f; long i;

    #pragma acc parallel for reduction(+: pi)
    for (i=0; i<N; i++)
    {
        double t=(double) ((i+0.5)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi=%f\n",pi/N);
    return 0;
}
```



# Exemplo: calculando PI com OpenACC

```
#include <stdio.h>
#define N 1000000000
int main(void) {
    double pi = 0.0f; long i;
    /* #pragma omp parallel for reduction(+: pi) OpenMP */
    /* #pragma acc parallel for reduction(+: pi) OpenACC */
    for (i=0; i<N; i++)
    {
        double t=(double) ((i+0.5)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi=%f\n",pi/N);
    return 0;
}
```



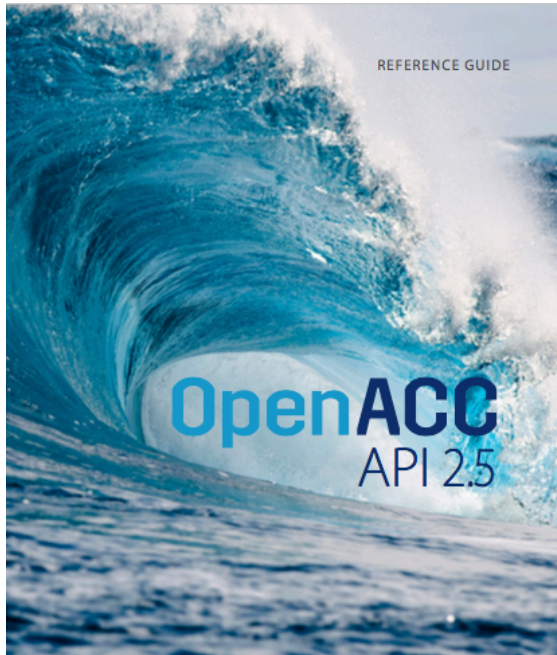


# Comparação do exemplo #0

Tipo	Sem diretivas	OpenMP(4)	Acelerador
Tempo (s)	6,118	2,238	0,351
Ganho (vezes)	---	2,73	17,43



# Padrão OpenACC



The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator device, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C and C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.

## General Syntax

C/C++  
**#pragma acc** directive [clause [,] clause...] new-line

## FORTRAN

**!\$acc** directive [clause [,] clause...] new-line

An OpenACC construct is an OpenACC directive and, if applicable, the immediately following statement, loop or structured block.

- “The OpenACC™ Application Programming Interface. Version 2.5a, October, 2015”

[http://www.openacc.org/sites/default/files/OpenACC\\_2pt5.pdf](http://www.openacc.org/sites/default/files/OpenACC_2pt5.pdf)

- Quick Reference Guide encontrado em:
- [http://www.openacc.org/sites/default/files/OpenACC\\_2.5\\_ref\\_guide\\_update.pdf](http://www.openacc.org/sites/default/files/OpenACC_2.5_ref_guide_update.pdf)
- [www.openacc.org](http://www.openacc.org)
  - NÃO DEIXEM DE VISITAR!!!!!!!!!!!!



EXAFLOP SISTEMAS

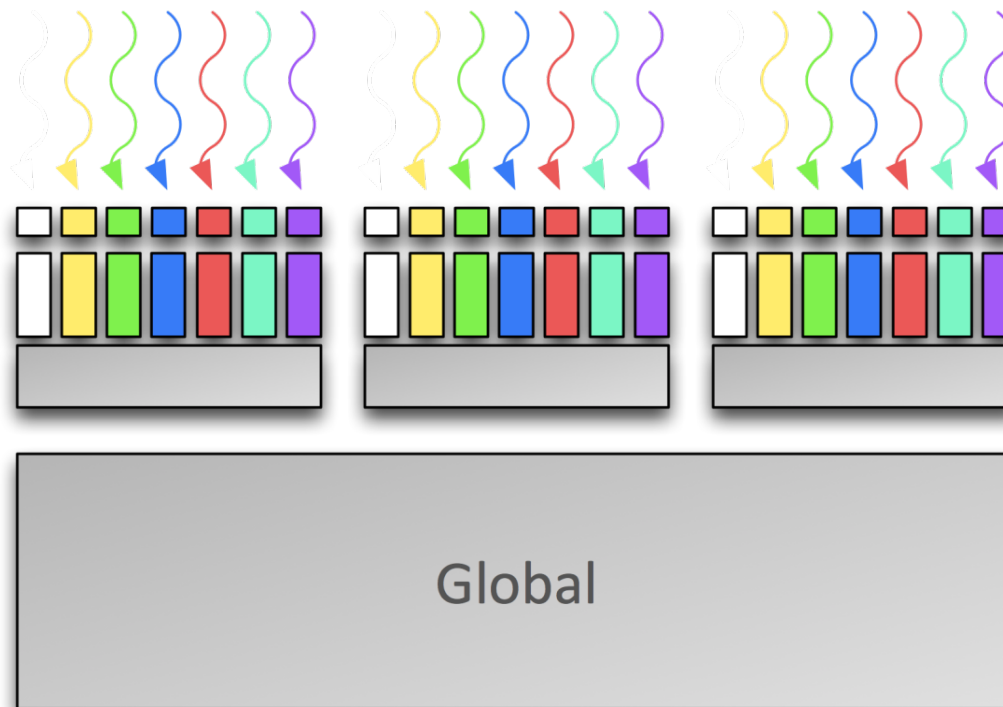
# Conceito básico: modelo de memória

- Acelerador não enxerga memória do host e vice-versa
- Dados devem ser enviados e recebidos, alocados e dealocados
  - Mas estas operações são em grande maioria implícitas, o compilador trata isso (mas não é obrigatório!)
- Não há coerência entre memória do host e acelerador
- Cópia da memória do host para a acelerador pode ser lenta
- No acelerador não há coerência entre threads
  - Race-conditions pode ocorrer
  - Programas que não tratam estes problemas são programas errados
- Alguns aceleradores possuem uma memória cache, rápida mas pequena
  - Ajuste fino do uso deste cache aumenta substancialmente o desempenho

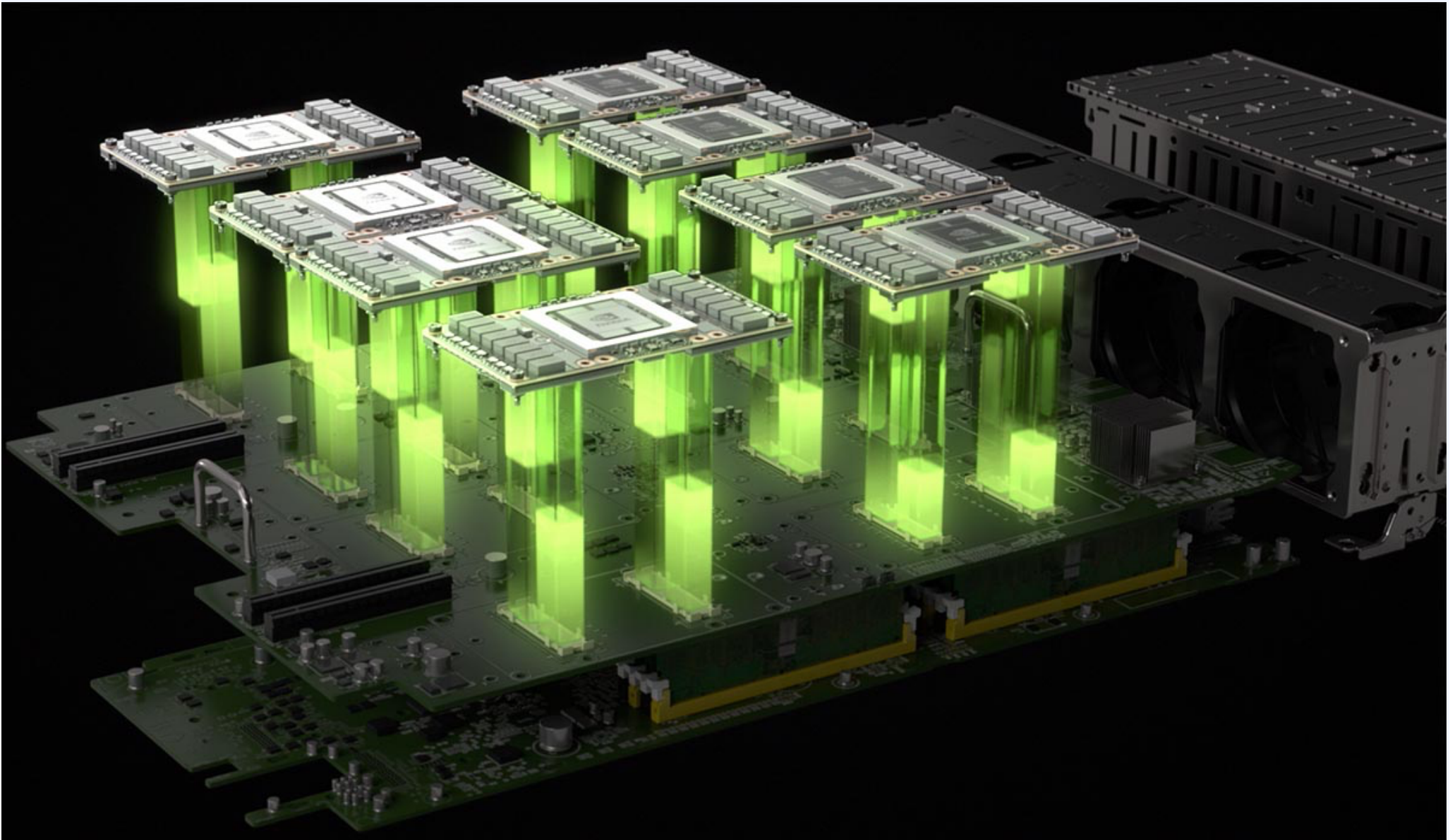


# Hierarquia de memória

- Memória local de uma *thread*
- Memória compartilhada no bloco de *threads*
- Memória global entre blocos de *threads*



# NVIDIA Architecture



EXAFLOP SISTEMAS

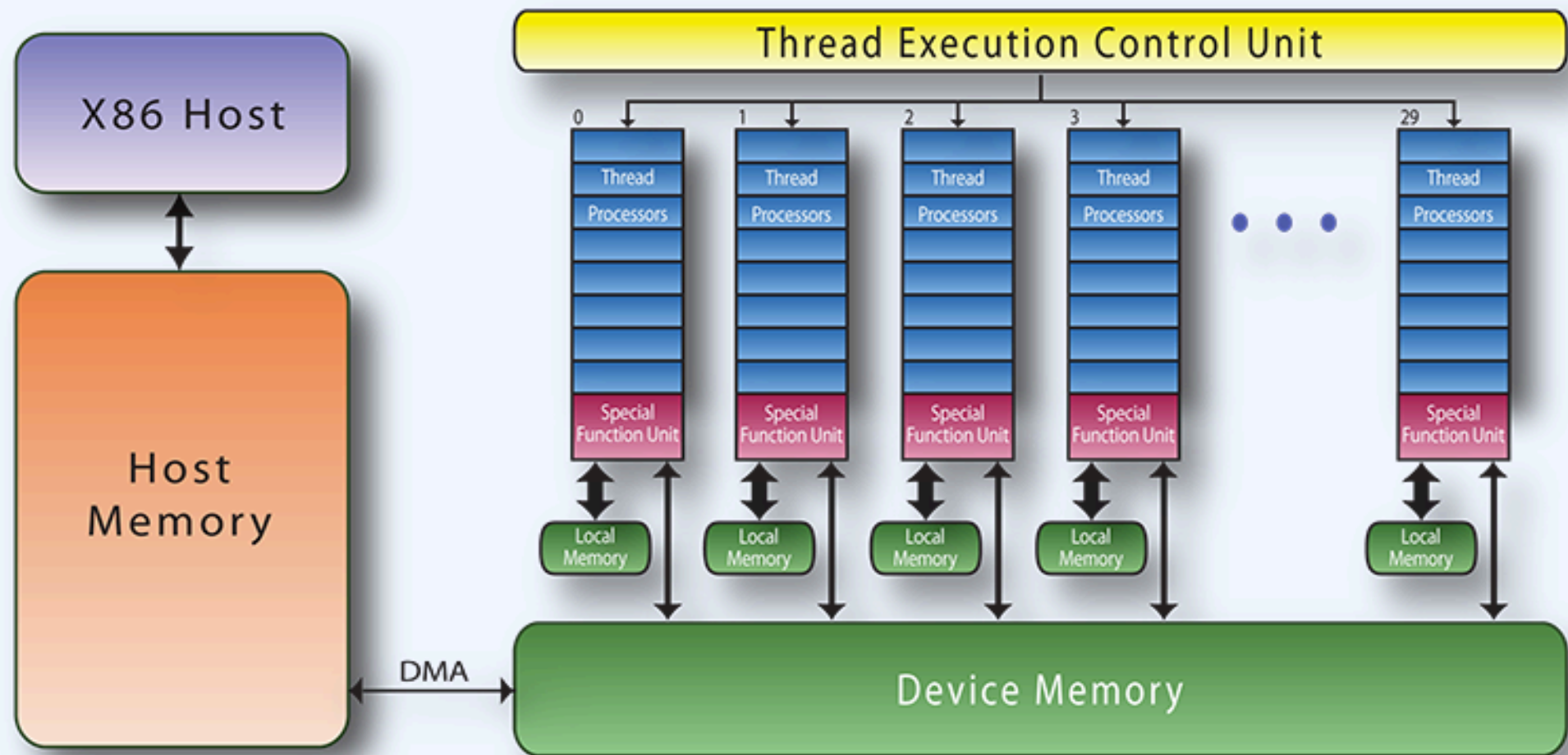
# Conceito básico: modelo de execução

- Grande parte do código é executado na host
- Regiões intensivas de cálculo são "offloaded", migrados para a host
  - Mas o programador deve explicitamente orientar o compilador, através das diretivas
- Trechos de código a serem acelerados criarão um ou mais "kernels"
  - Denominação para um conjunto de instruções de cálculo na GPU
  - Kernel pode ser um simples laço, ou até mesmo um conjunto de kernels pode descrever uma subrotina ou função
- Mesmo em regiões intensivas o host também trabalhará, tomando conta da execução dos kernels, migração de dados, cópias, sincronismos, inicialização e finalização etc
- Um acelerador lança paralelismo em *gangs*, cada *gang* possui um ou mais *workers* sendo que em um *worker* poderão ocorrer operações vetoriais denominadas *vector*
  - Hãã???????





# NVIDIA Architecture



**The Portland Group<sup>®</sup>**

Accelerator Overview Demo  
©2009 The Portland Group, Inc.



EXAFLOP SISTEMAS

# Sintaxe básica (C e Fortran)

```
#pragma acc nome_diretiva [cláusula [,cláusula]...]
    bloco estruturado de código
```

-----

```
!$acc nome_diretiva [cláusula [,cláusula]...]
    bloco estruturado de código
!$acc end nome_diretiva
```





# As diretivas

1. **parallels**: executa de forma paralela e íntegra o bloco no acc.
2. **kernels**: cria e computa kernels (região paralela distribuída) no acc.
3. **data**: alocação e movimentação de dados
4. **enter data**: alocação e movimentação de dados até o fim do programa
5. **exit data**: movimentação e desalocação de dados alocados no **enter data**
6. **host\_data**: endereça no host dados do acc. (ponteiro do target)
7. **loop**: descreve paralelismo do laço no acc.
8. **cache**: especifica dados para cache do acc.
9. **declare**: aloca dados no acc.
10. **atomic**: assegura que o bloco deve ser executado atomicamente no acc.
11. **update**: atualiza dados no acc. e/ou no host
12. **wait**: espera finalização de operação assíncrona
13. **routine**: cria rotina com código do acc. e expõe nome para o host
14. (link, executable, API com outras arquiteturas, procedure calls...)



# Limitações importantes

- Não podem existir chamadas de rotinas dentro de regiões aceleradas
  - Solução: inline manual ou automático
  - Se for automático e estiver em outros arquivos-fonte, vai ser necessário utilizar -ipa (ler documentação)
  - Solução #2: segundo Michael Wolf utilizar PGI 2014 em diante
    - Tentei testar na dinâmica de um modelo atmosférico
- Variáveis derivadas de módulos (array de um elemento de um tipo em Fortran) não podem ser copiados para o acelerador
  - As vezes o compilador precisa saber o “formato” do *array*
  - Esta operação se chama “*deep-copy*” e parece ter *overhead*
  - Solução para *arrays* complicados: cópia de memória



# Dicas

- Laços aninhados são os melhores para serem acelerados
  - Aproveitam a hierarquia de memória
  - Aproveitam os níveis de paralelismo do acelerador
- Sobreposição de comunicação com computação é possível e indicado
  - As regiões podem ser síncronas ou assíncronas
    - Ver diretiva “wait”



# Dicas

- Iterações dos laços devem ser independentes
  - “Laços triangulares” não podem ser acelerados
- Ajude o compilador: use a chave “restrict” ou “independent” do C
- Compiladores podem se perder com o gerenciamento do que enviar/receber ao acc.
- Não utilizar aritmética de ponteiros
- Use memória contígua para arrays multi-dimensionais
  - E o percorra de acordo com sua construção na memória
    - C é diferente de Fortran!!!



# Algumas conclusões

- OpenACC torna fácil o trabalho de acelerar trechos de código
- Operadores básicos: alto nível
  - Fácil de entender e programar
  - O OpenMP também é alto nível, e é robusto!
- Há sim ganho de desempenho
  - Pelo menos nos testes simples
  - Demais testes: ver GTC2012 da NVIDIA, GTC2013, SC13 e site [www.openacc.org](http://www.openacc.org)



# Fontes importantes

- [www.openacc.org](http://www.openacc.org)
- [developer.nvidia.com/openacc](http://developer.nvidia.com/openacc)
- [gcc.gnu.org/wiki/OpenACC](http://gcc.gnu.org/wiki/OpenACC)
- [www.researchgate.net/project/OpenACC-2-support-on-GCC-61-Early-experiences](http://www.researchgate.net/project/OpenACC-2-support-on-GCC-61-Early-experiences)
- <https://www.pggroup.com/resources/accel.htm>
- <http://a.co/c7OFtUD> (Parallel Programming with OpenACC de Rob Farber)



# Parte 2: COMPILADORES



# Compilador PGI

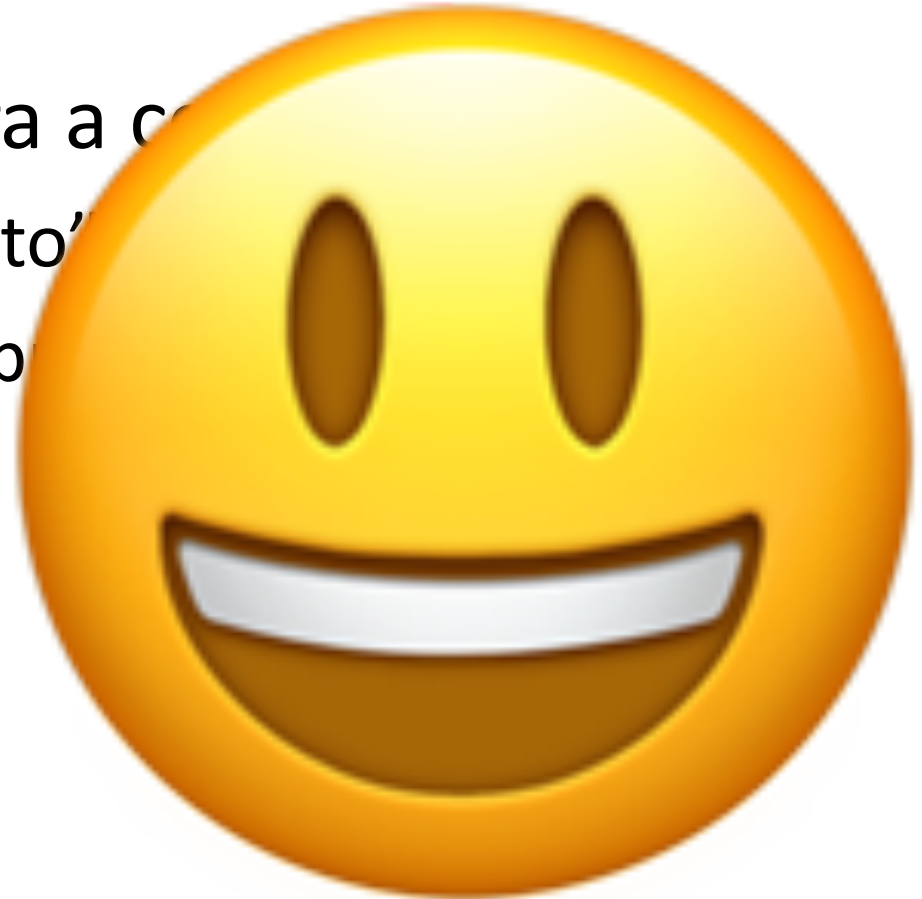
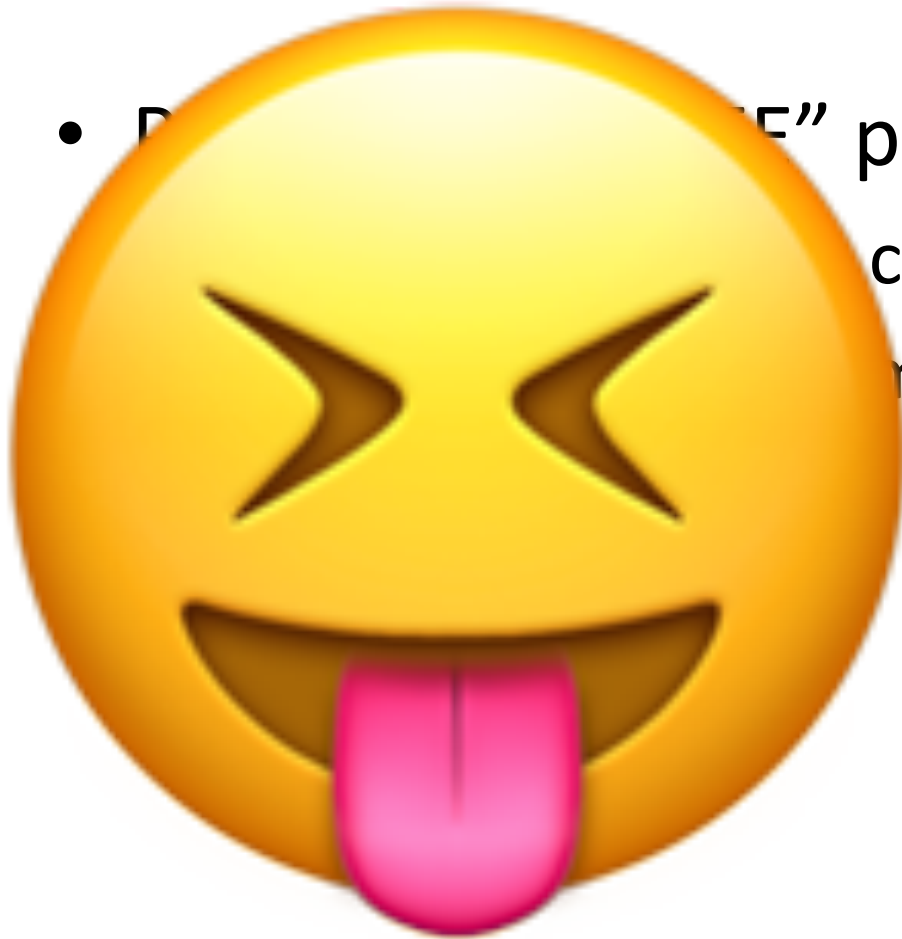
- Ótimo compilador PAGO
- Não precisa do CUDA Toolkit para instalação, mas precisa dos drivers da Nvidia
- Referência em acelerar códigos
  - Tanto é que a NVIDIA comprou a PGI
- Linux, Windows, Mac OSX
- Precisa ser administrador para instalação
- Valores (fev/2017): US\$ 1.999 licença "flutuante" ou US\$ 1.399 licença "node-locked"
  - 50% de desconto para academia
- [www.pggroup.com](http://www.pggroup.com)





# Novidade boa

- D... "E" para a c... custo' n/p



# Instalação do PGI (passo-a-passo)

- Ir no site
- Fazer o download do arquivo para a sua arquitetura (99% de ser Linux x86-64)
- Criar um diretório (ex, pgi-inst) e entrar nele
- Descompactar arquivo obtido
- Executar o script de instalação ./install



# Instalação do PGI

- Quando perguntado, escolher "Single system install" (opção 1)
- O diretório de instalação deve ser um que seu usuário possa escrever (ex. /home/usuario/pgi)
- Ir confirmando, confirmando, confirmando
- Quando perguntado pela licença, digitar opção 4 (I'm not sure (quit now and re-run this script later,))
- Confirmar até terminar



# Instalação do PGI

- Após instalação, setar seu PATH e seu LD\_LIBRARY\_PATH
- `export PATH=/home/usuario/pgi/linux86-64/2017/bin:$PATH`
- `export LD_LIBRARY_PATH=/home/usuario/pgi/linux86-64/2017/lib:$LD_LIBRARY_PATH`
- Testar (com os exemplos deste curso...)



# Compilador Cray

- Pertencente ao Cray Linux Environment (CLE)
- ~~Responsável por aglutinar e publicar o padrão OpenACC~~
  - (era: o cara que fazia isso foi trabalhar na Nvidia!)
  - O compilador próprio tem suporte total ao padrão
  - Muito mais “amigável” no tratamento de regiões aceleradas do código
- Um compilador Cray só existe em um Cray



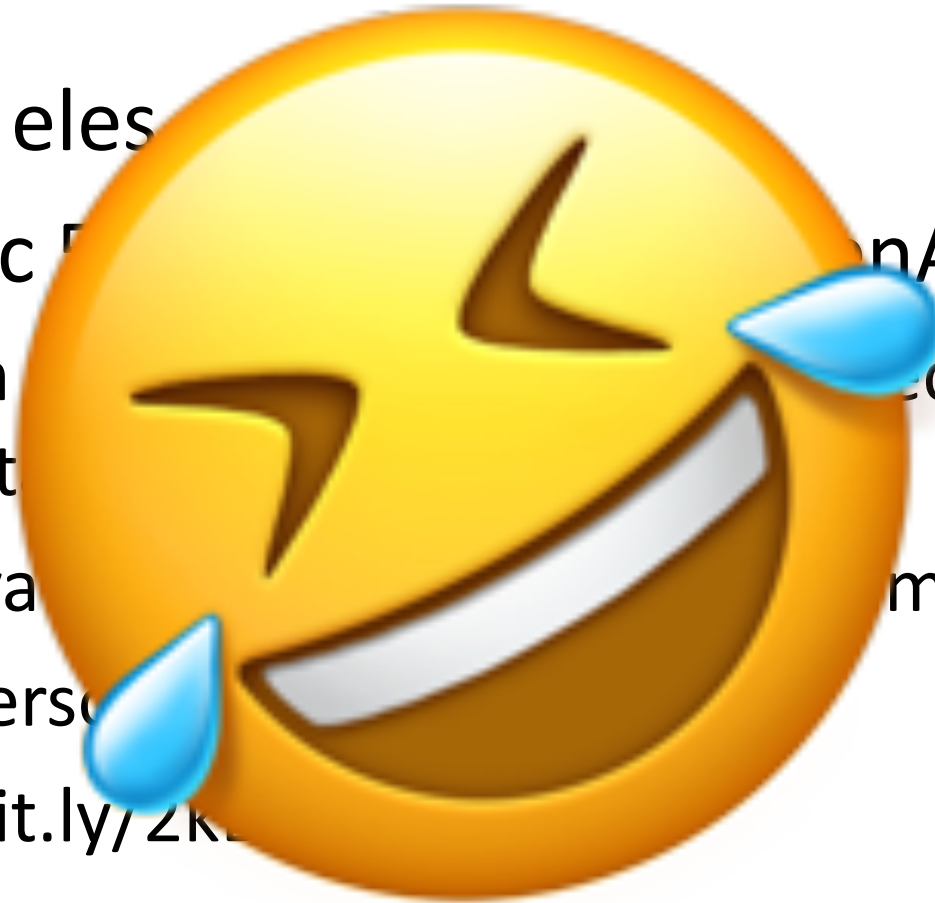
# GCC+OpenACC (slide de 2015)

- PathScale (☺) é pago e não testei (☹)
- GCC (isso sim é uma boa notícia!)
  - A “Mentor Embedded” está fazendo esta herculana tarefa
  - Dizem que é previsto para o 5.0
  - *OpenACC support has been merged into GCC 5, but a handful of patches are still pending that are needed for nvptx offloading, so that's not yet functional.* (visto em <https://gcc.gnu.org/wiki/OpenACC>)
  - Escutei sobre ele no SC13, mas não no SC14



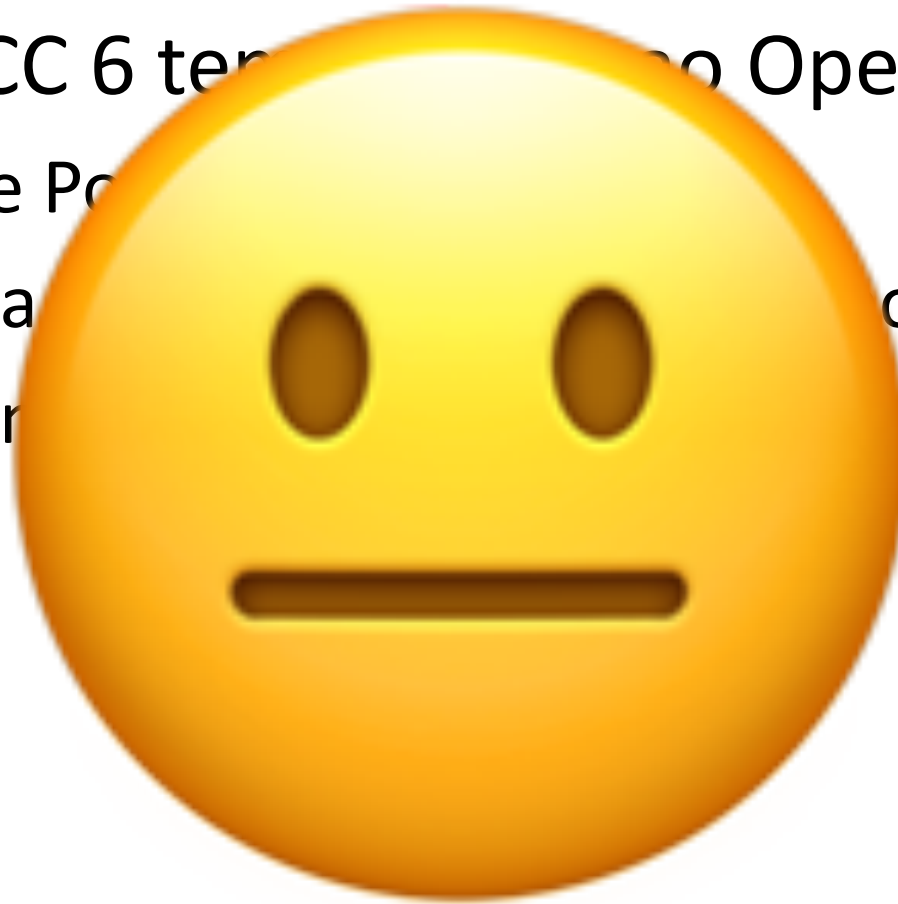
# GCC+OpenACC

- Adivinha: eles
- Status: gcc 5.4.0 + OpenACC 1.0
  - Não tem `host_data`
  - A diretiva `wait` não é implementada
  - Tem diversos bugs
  - <http://bit.ly/2k1>



# GCC+OpenACC

- Status: GCC 6 tem suporte ao OpenACC 2.0a
  - x86\_64 e PowerPC
  - A diretiva `wait` não é suportada
  - E o desempenho é ruim





# Outros compiladores (2017)

- accULL: Universidad de La Laguna/EPCC
  - Funciona, compila códigos simples, é um “source to source”, convertendo trecho em OpenACC para CUDA, usa Python no meio do caminho...
- Omni: RIKEN/University of Tsukuba (Japão)
  - Melhorou bastante com relação a 2015
  - Teste com o pi: 0,787 segundos (excelente!)
  - Não foram feitos mais testes...

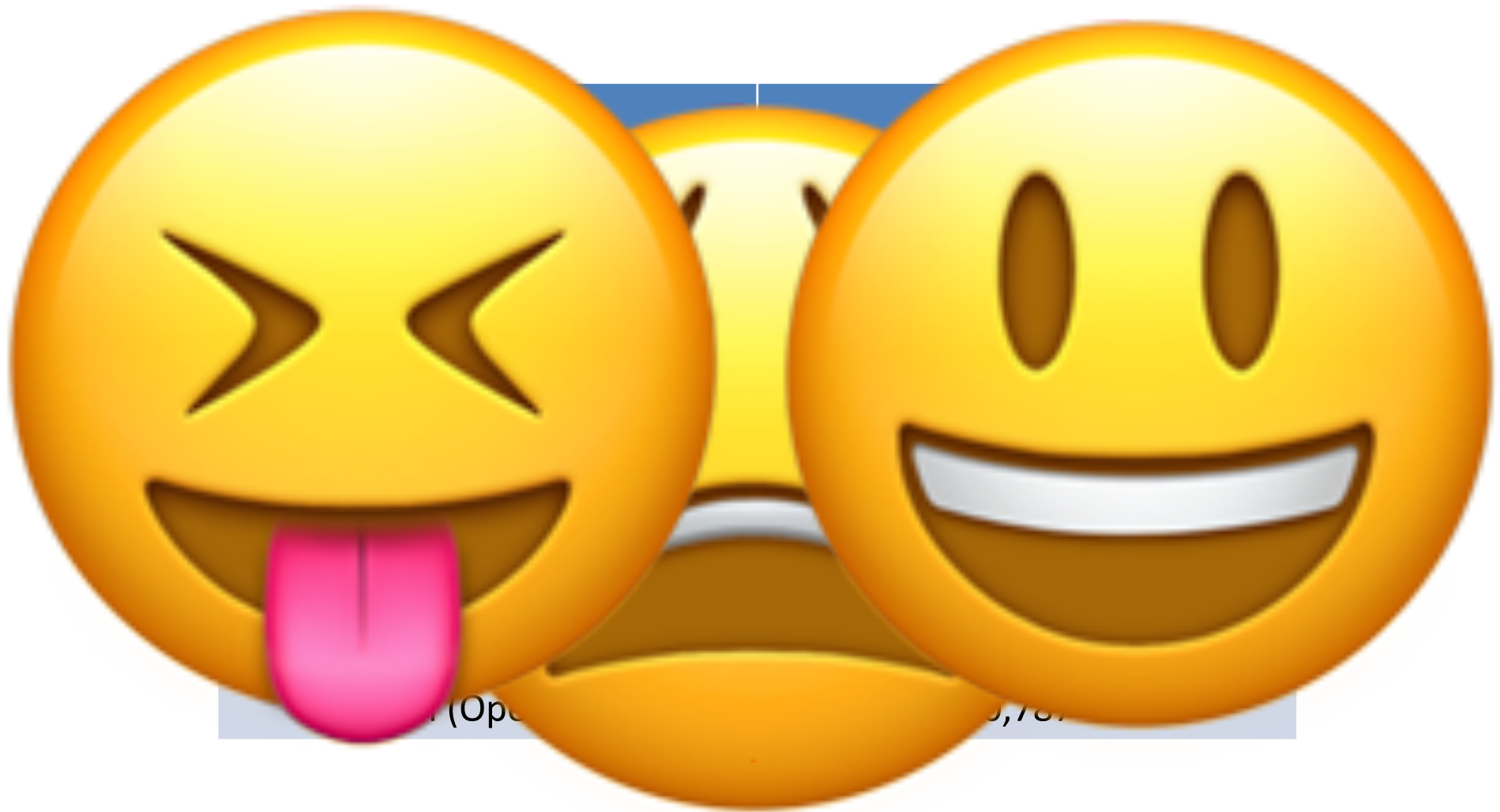


# Outros compiladores

- RoseACC: University of Delaware/LLNL
  - Tem que fazer fork do github, outro source-to-source, não testei
- OpenUH: University of Houston
  - Beeeem complicando para instalar, é gigante, tem suporte alpha a OpenACC mas suporta Coarray Fortran, existe a anos
- OpenARC: Oak Ridge Nat. Lab.
  - Bastante científico, quem quiser se aventurar...



# Teste rápido do pi.c



# Parte 3: PRÉ-PRÁTICA

- Ambiente
- Flags de compilação e mensagens
- Execução do programa
- Medição de tempo total e por região acelerada



# Ambiente

- Na sua conta do SDUMONT, carregar o módulo PGI  
module load PGI/compilers-16.5
- Copiar os arquivos do curso  
`cp ~professor/pedro.lopes/MP11-OpenACC.tar.gz .`
- Descompacte-o!



# Quem quiser levar pra casa

[exaflop.com.br/MP11-OpenACC-2017.tar.gz](http://exaflop.com.br/MP11-OpenACC-2017.tar.gz)

(apagarei em uma semana!)



EXAFLOP SISTEMAS



# Entendendo o pacote

• mp11-acc2017.pdf	Esta apresentação
• e1-pi.c	Cálculo do PI para acelerar
• e2-laplace2d.c	Jacobi para acelerar
• e3-laplace2d.c	Jacobi para otimizar
• e4-axpy.c	SAXPY para acelerar
• roda.srm	Arquivo de submissão
• solucao	Diretório de soluções
• - s1-pi.c	Solução do e1
• - s2-laplace2d.c	Solução do e2
• - s3-laplace2d.c	Solução do e3
• - s4-axpy.c	Solução do e4



# Entendendo o ambiente

- Rodar o dados-ambiente.srm  
    sbatch dados-ambiente.srm
- Verificar o que ele reportou
- Variável de ambiente ACC\_DEVICE\_NUM controla qual acelerador será usado por padrão
  - Sua execução estará sozinha no nó: não há necessidade desta variável
- ACC\_NOTIFY irá notificar quando executar uma região acelerada
  - Ótimo para debug



# Recordando

- Submeter: `sbatch ./rodar.srm`  
(Vai retornar um ID: lembre-o!)
- Investigar a saída: `less slurm-ID.out`
- Antes de submeter lembrar de modificar o nome do executável dentro do `rodar.srm`



## Parte 3: PRÁTICA

# Hands-on!



# Exercício #1 (faremos juntos!): e1-pi.c

- Objetivo: compilar, executar e medir tempo de execução do e1-pi para comparar execução sequencial, OpenMP e OpenACC
- Conceitos: paralelismo, speed-up, medição confiável de tempo
- Anotar a fórmula:

$$S_p = \frac{T_1}{T_p}$$

$S \Rightarrow$  Speed-up

$p \Rightarrow$  número de processadores

$T \Rightarrow$  Tempo



# Passos!

- Abrir o arquivo com algum editor
- Entender o laço
  - Ele pode ser executado em paralelo?
- Compilar!
- Executar e cronometrar
- Anotar
- Calcular speed-up



# O e1-pi.c

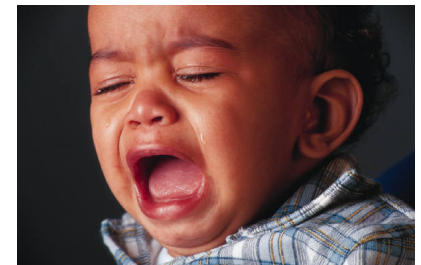
```
#include <stdio.h>
#include <omp.h>
#define N 1000000000
int main(void) {
    double pi = 0.0f; long i;
    printf("Numero de processadores = %d\n", omp_get_max_threads());
    #pragma omp parallel for reduction(+: pi) /* OpenMP */
    #pragma acc parallel loop reduction(+: pi) /* OpenACC */
    for (i=0; i<N; i++)
    {
        double t=(double) ((i+0.5)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi=%f\n",pi/N);
    return 0;
}
```





# Compilação “na mão”

- Receita
  - Compilador: “ pgc++ ”
  - Chave para criação do executável: “ -o e1-pi ”
  - Chave para ligar OpenMP: “ -mp ”
  - Chave para ligar OpenACC: “ -acc ”
  - Chave para escolher o acelerador: “ -ta=nvidia ”
  - Saber o que ele está fazendo: “ -Minfo ”
  - Forma de rodar: “ ./e1-pi ”
- ATENÇÃO: não utilize -mp e -acc junto!
  - A máquina explodirá e criará um buraco-negro



# Mas tem Makefile!!!

- Digite “make” e as opções mostram o que pode ser feito
- Alguns tem OpenMP implementado, outros não, mas se quiser implementar OpenMP fiquem a vontade
- make clean limpa tudo!

\* Aos fortes de coração o Makefile tem “easter egg”

# Medir tempo

- Simples, muito simples

`/usr/bin/time -p ./e1-pi-omp` (o roda.srm tem!)

- Dica: com a chave “-mp” para controlar o número de processadores use a variável OMP\_NUM\_THREADS (ver o roda.srm!)

`/usr/bin/time -p OMP_NUM_THREADS=4 ./e1-pi-omp`



# Medir tempo com acelerador

- Também é muito simples
  - Igual ao OpenMP
- Para assegurar que o acelerador está sendo usado modifique a variável ACC\_NOTIFY

```
/usr/bin/time -p ACC_NOTIFY=1 ./e1-pi-acc
```



# Resultados

- São compatíveis com o apresentado?

Tipo	Sem diretivas	OpenMP(4)	Acelerador
Tempo (s)	6,118	2,238	0,351
Ganho (vezes)	---	2,73	17,43



# Diretiva utilizada: parallel

- A `parallel` criou uma seção paralela no bloco estruturado seguinte, lançando vários *gangs*
  - Todas as *threads* executam a mesma coisa redundantemente!
- Lançou vários *gangs* para as iterações do laço
- Como havia a outra diretiva “`loop`” logo a seguir quebrou o espaço de índices em pedaços e os distribuiu nos *gangs*
- A cláusula “`reduction`” realizou a soma dos *gangs*



## Exercício #2: e2-laplace2d.c

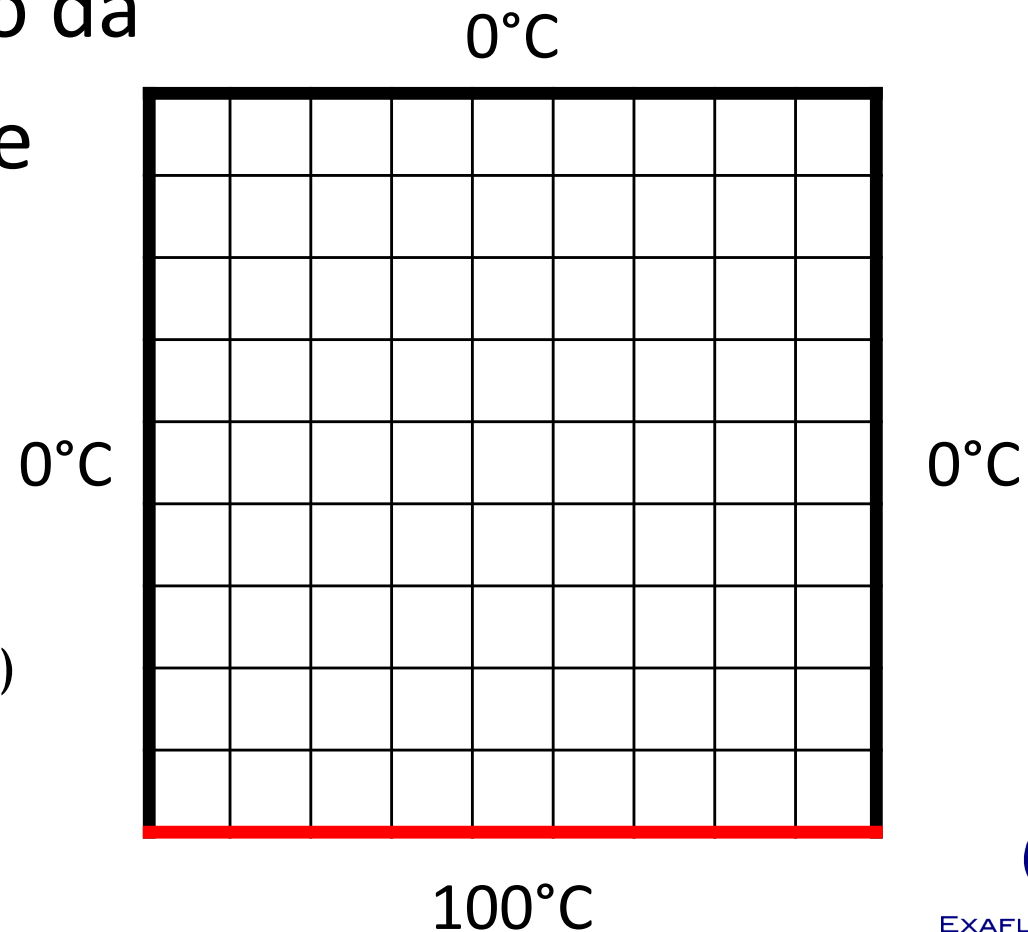
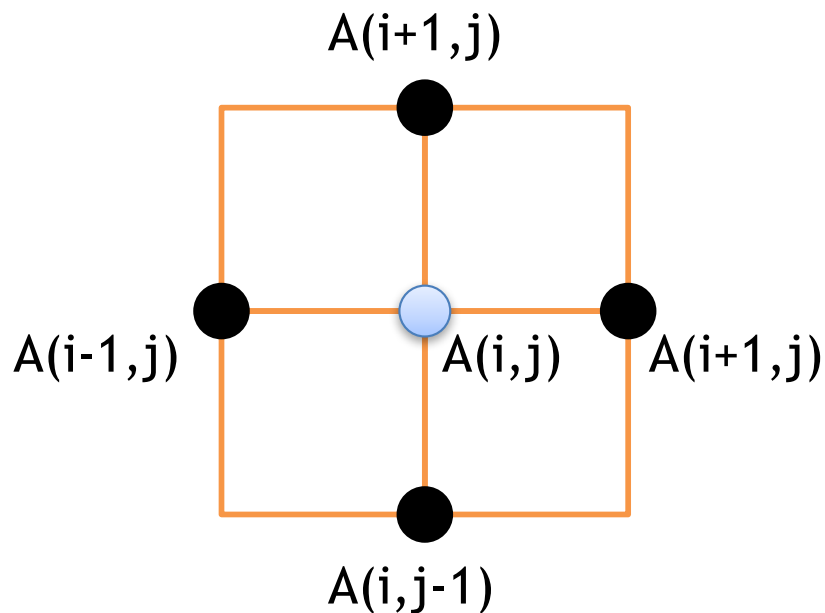
- Objetivo é acelerar um programa mais complexo, com dois laços e um “while”
- Ele já está paralelizado com OpenMP
- Tarefa: encontrar a diretiva e implementar





# O problema

- Converge iterativamente para um valor a partir da média dos pontos vizinhos
- Exemplo: solução da Equação de Laplace



# Diretiva a utilizar: kernels

- A “kernels” cria um ou mais *kernels* no acelerador para executar laços eficientemente
- Utiliza bem os graus de paralelismo existentes
- O laço mais externo é completado antes da execução do laço mais interno
- É rápido, mas não tem as mesmas operações do “parallel”
  - VER NO PADRÃO AS DIFERENÇAS



# Diretiva kernels ("foto" do RefGuide)

## Kernels Construct

A **kernels** construct surrounds loops to be executed on the device, typically as a sequence of kernel operations.

### C/C++

```
#pragma acc kernels [clause [,] clause...] new-line  
{ structured block }
```

### FORTRAN

```
!$acc kernels [clause [,] clause...]  
structured block  
!$acc end kernels
```

Compute Construct and Data clauses are also allowed; data clauses on the kernels construct modify the structured reference counts for the associated data.



# Solução

```
while ( error > tol && iter < iter_max ) {  
    error=0.0;  
    #pragma omp parallel for shared(Anew, A)  
  
    for( int j = 1; j < n-1; j++) {  
        for(int i = 1; i < m-1; i++) {  
            Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);  
            error = max(error, abs(Anew[j][i] - A[j][i]));  
        }  
    }  
    #pragma omp parallel for shared(Anew, A)  
  
    for( int j = 1; j < n-1; j++) {  
        for( int i = 1; i < m-1; i++ ) {  
            A[j][i] = Anew[j][i];  
        }  
    }  
    iter++;  
}
```



# Solução

```
while ( error > tol && iter < iter_max ) {  
    error=0.0;  
    #pragma omp parallel for shared(Anew, A)  
    #pragma acc kernels  
        for( int j = 1; j < n-1; j++) {  
            for(int i = 1; i < m-1; i++) {  
                Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);  
                error = max(error, abs(Anew[j][i] - A[j][i]));  
            }  
        }  
    #pragma omp parallel for shared(Anew, A)  
    #pragma acc kernels  
        for( int j = 1; j < n-1; j++) {  
            for( int i = 1; i < m-1; i++ ) {  
                A[j][i] = Anew[j][i];  
            }  
        }  
    iter++;  
}
```



# Que péssimo desempenho!

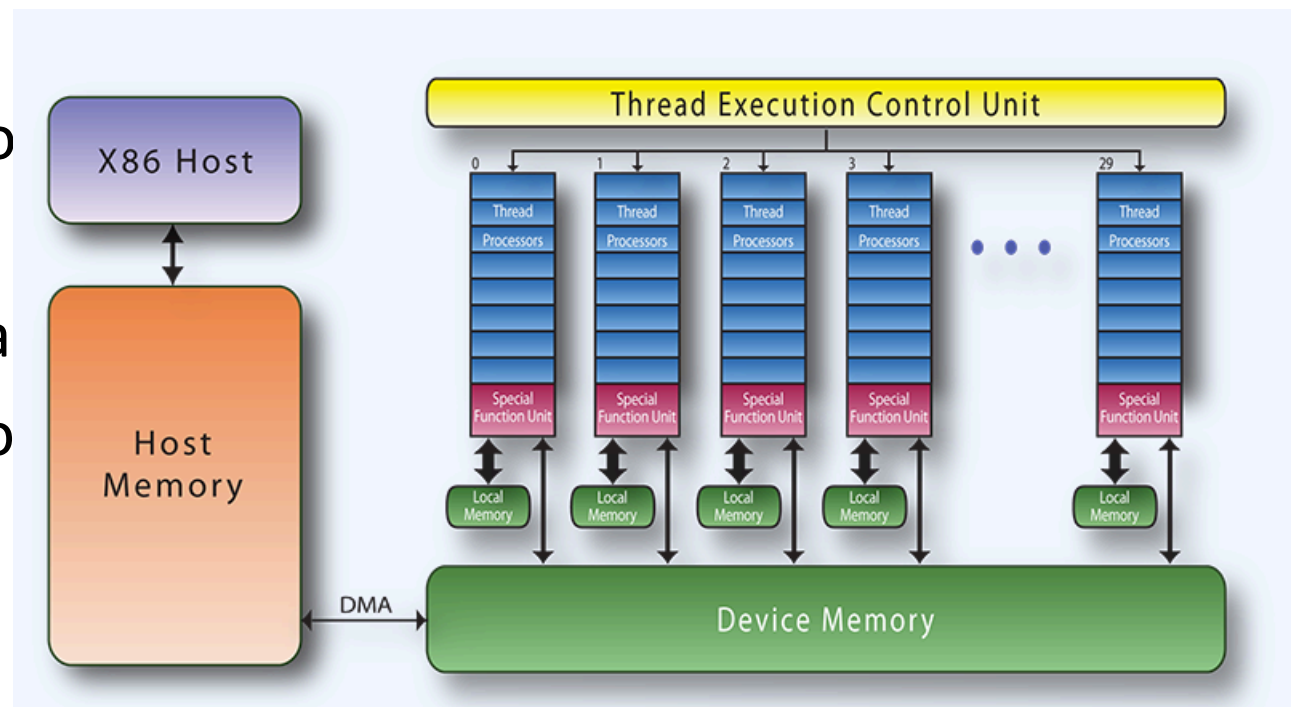
- Compare os tempos com e sem acelerador
- Onde está o problema?



# Compilador fez o melhor que pode

- Mas ele NÃO pode desrespeitar o código

1. Enquanto condicional do while for satisfeita...
2. copia A para a placa
3. Computa primeiro laço
4. Copia Anew para CPU
5. Copia Anew para placa
6. Computa segundo laço
7. Copia A para a CPU
8. Volta para 1.





## Exercício #3: e3-laplace2d.c

- Objetivo é otimizar a execução do exercício #2
- Ele já está acelerado
- Verifique a problemática
  - Entenda “no código” e “no algoritmo” o problema
- Este exercício pode ser melhorado ainda mais



# Diretivas a utilizar: data

- A diretiva “data” diz ao compilador, com suas cláusulas
  - copy(vetor): Copie um vetor para o acelerador e o traga de volta no fim
  - copyin(vetor): Copie um vetor para o acelerador
  - copyout(vetor): Copie um vetor para o *host*
  - create(vetor): Criar um vetor no acelerador mas não o inicializa com dados

(lembrar:

```
#pragma acc nome_diretiva [cláusula [,cláusula]...]
      bloco estruturado de código )
```



# Perguntas interessantes

- A matriz A precisa estar no acelerador?
  - Ela está criada lá?
  - Precisa ser inicializada?
  - Precisa dela no fim dos cálculos?
- A matriz Anew serve para quê?
  - Precisa dela no acelerador?
  - Precisa dela no host?
- Traduzir a resposta destas perguntas em operações de cópia das matrizes “dê” e “para” o acelerador



# Solução

```
while ( error > tol && iter < iter_max ) {  
    error=0.0;  
    #pragma omp parallel for shared(Anew, A)  
    #pragma acc kernels  
        for( int j = 1; j < n-1; j++) {  
            for(int i = 1; i < m-1; i++) {  
                Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);  
                error = max(error, abs(Anew[j][i] - A[j][i]));  
            }  
        }  
    #pragma omp parallel for shared(Anew, A)  
    #pragma acc kernels  
        for( int j = 1; j < n-1; j++) {  
            for( int i = 1; i < m-1; i++ ) {  
                A[j][i] = Anew[j][i];  
            }  
        }  
    iter++;  
}
```



# Solução

```
#pragma acc data copy(A), create(Anew)
while ( error > tol && iter < iter_max ) {
    error=0.0;
    #pragma omp parallel for shared(Anew, A)
    #pragma acc kernels
        for( int j = 1; j < n-1; j++) {
            for(int i = 1; i < m-1; i++) {
                Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
                error = max(error, abs(Anew[j][i] - A[j][i]));
            }
        }
    #pragma omp parallel for shared(Anew, A)
    #pragma acc kernels
        for( int j = 1; j < n-1; j++) {
            for( int i = 1; i < m-1; i++ ) {
                A[j][i] = Anew[j][i];
            }
        }
    iter++;
}
```



# Possível melhora!

- Tirar os dois kernels
- Colocar um “parallel” com reduction
- Especificar os loops
- Mudar o número de gangs, workers e vectors



## Exercício #4: e4-axpy.c

- Objetivo e acelerar o AXPY e MANTER no acelerador o dado entre execuções da função
- Utilizar os dados da primeira execução na segunda execução
- Serão utilizadas algumas diretivas de gerenciamento de dados
  - Vide exercício anterior



# Diretiva a utilizar: update

- A “update” atualiza o conteúdo de um vetor no acelerador ou no *host* dependendo de suas cláusulas
  - device(vetor): do *host* para o acelerador
  - host(vetor): do acelerador para o *host*





# O trecho acelerado deve saber que as memórias estão lá!

- Incluir, na diretiva que acelerou o laço dentro da AXPY, a cláusula “present”
  - Ela informa que as memórias estão lá e não precisa ser enviada
  - Caso contrário o “parallel” irá criar automaticamente um  
“copy(x,y)”



# Obrigado!!!

[www.exaflop.com.br](http://www.exaflop.com.br)

[pedro.lopes@exaflop.com.br](mailto:pedro.lopes@exaflop.com.br)



EXAFLOP SISTEMAS