ERAD-SP 2017
Tutorial de OpenACC

Pedro Pais Lopes

08/04/2017

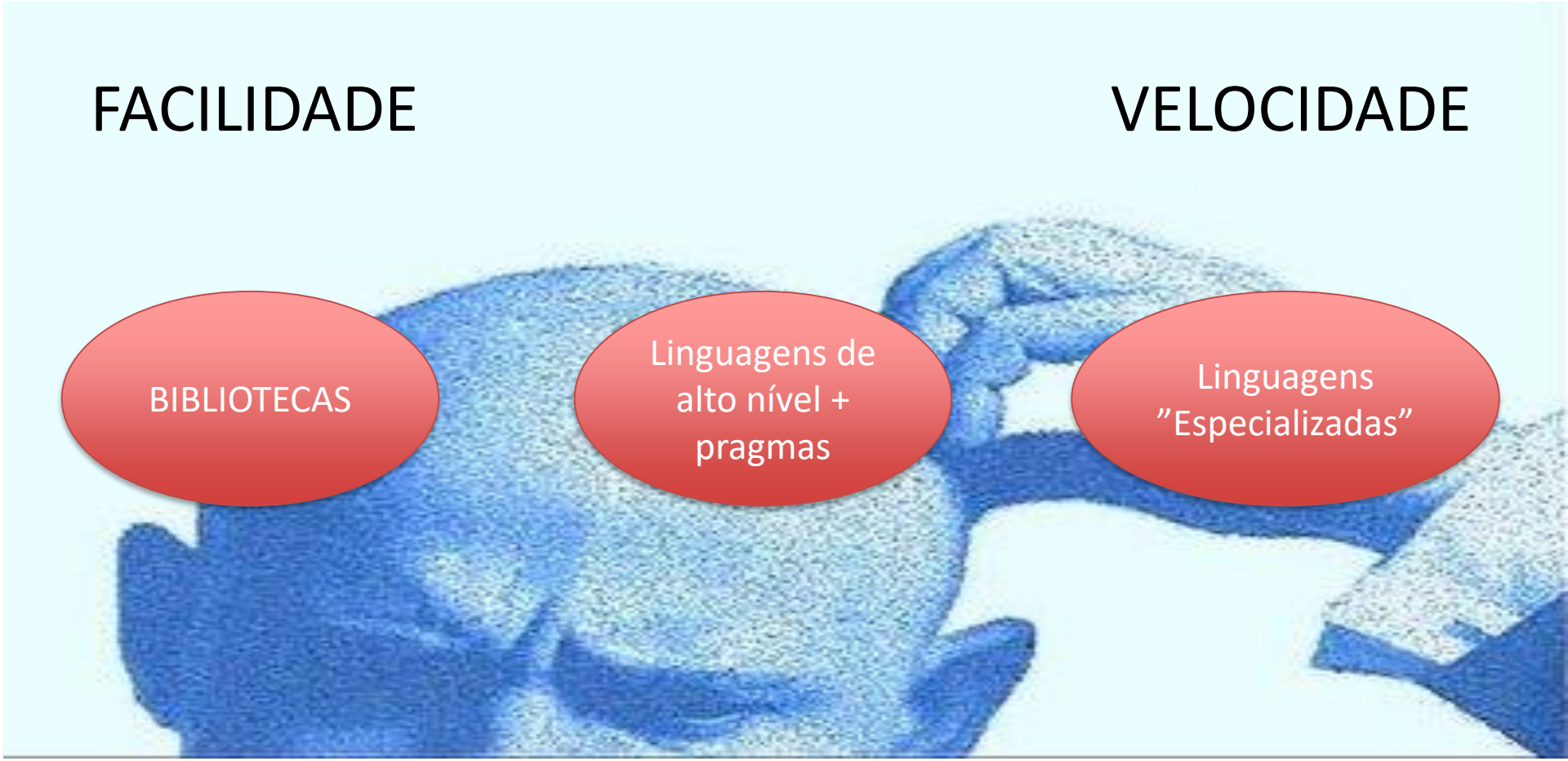
OpenACC?

- O que é?
- Para que serve?
- Como uso?
- O que preciso?
- Vai me ajudar?
- Onde consigo mais informação?

Ponto de largada!

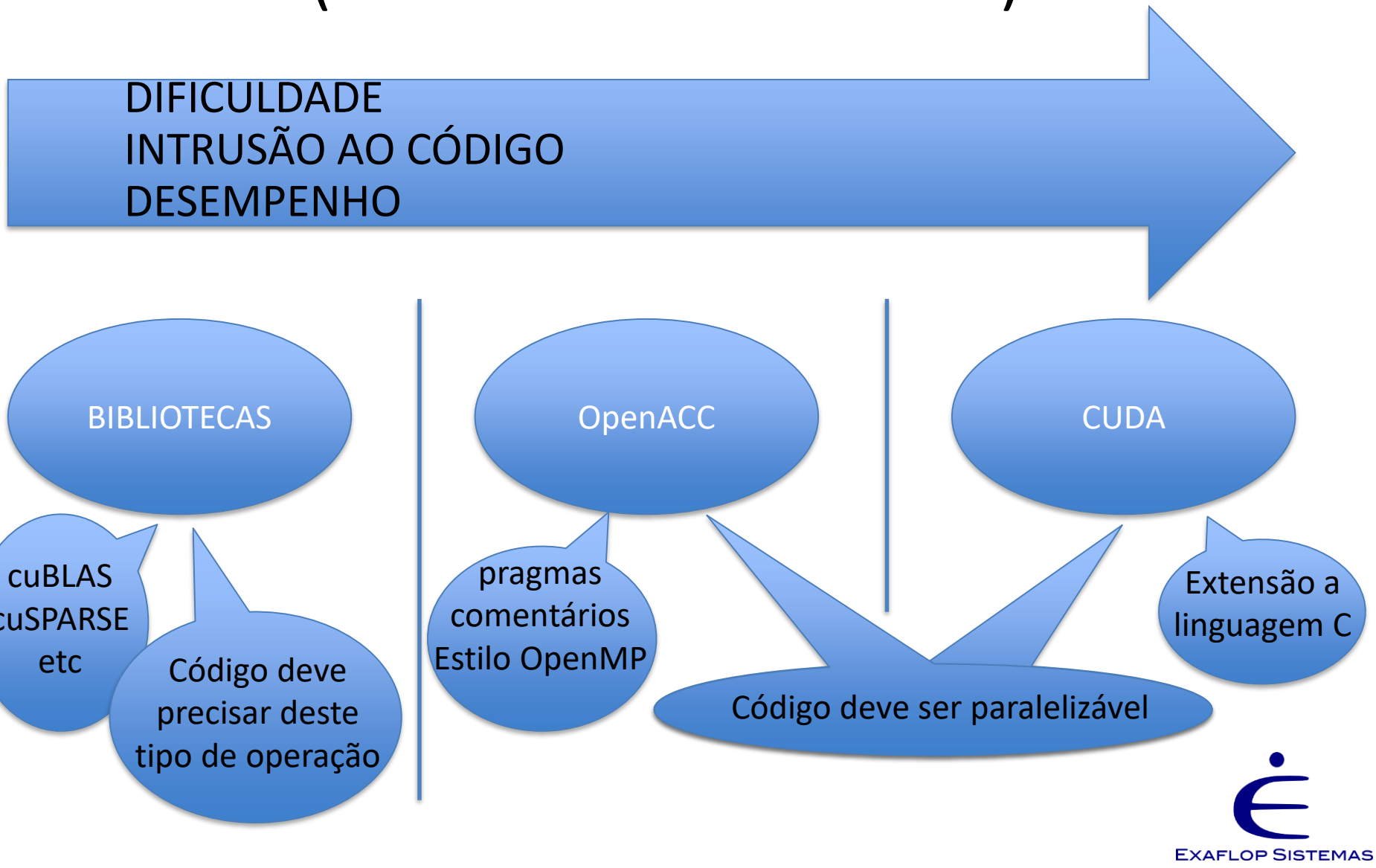
EXAFLOP SISTEMAS

Quero acelerar meu código, o que escolho?



EXAFLOP SISTEMAS

Depende do seu grau de liberdade (em todos os sentidos...)



EXAFLOP SISTEMAS

O acelerador

- Acelerador: dispositivo que realiza cálculos matemáticos massivos, muito específico e externo ao conjunto CPU-Memória

- Em geral a inequação abaixo é respeitada

$$nCores_{acelerador} >>> nCores_{CPU}$$

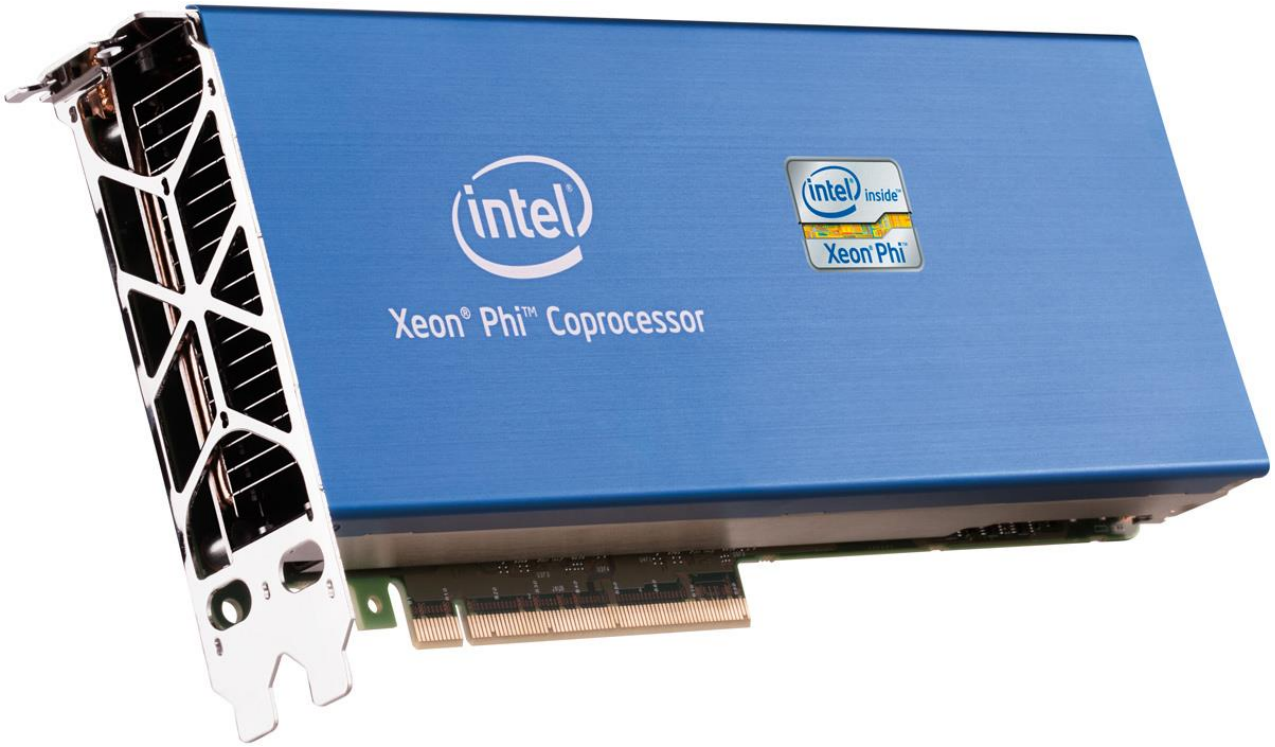
EXAFLOP SISTEMAS

Exemplo

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

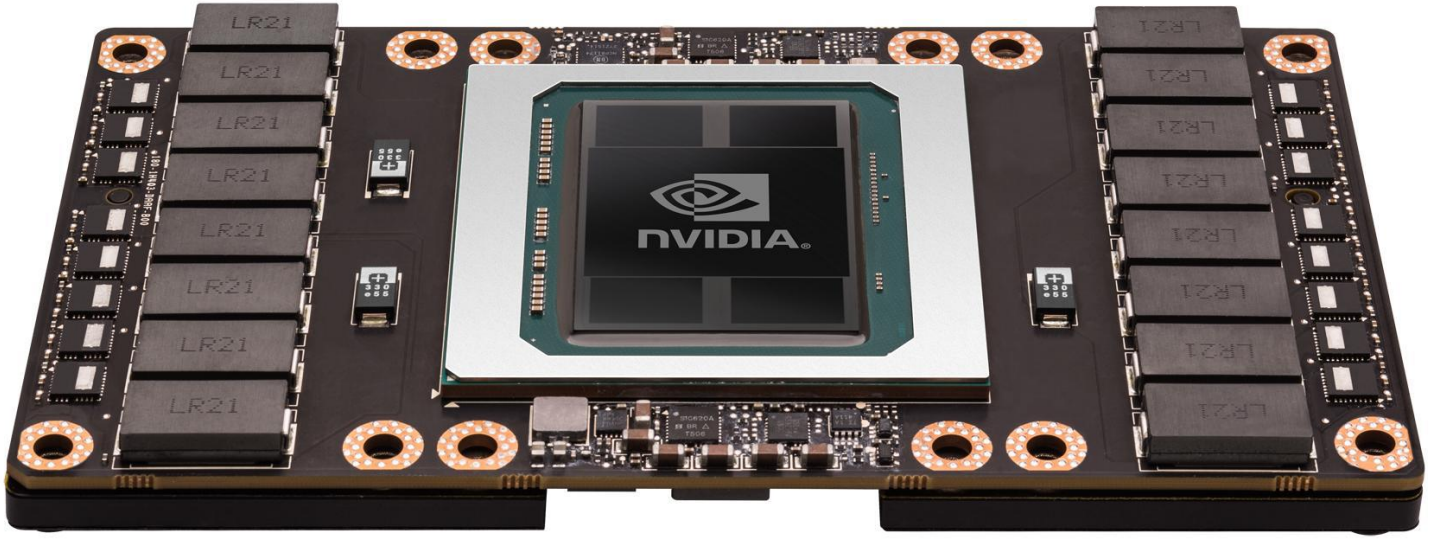
EXAFLOP SISTEMAS

Outro exemplo



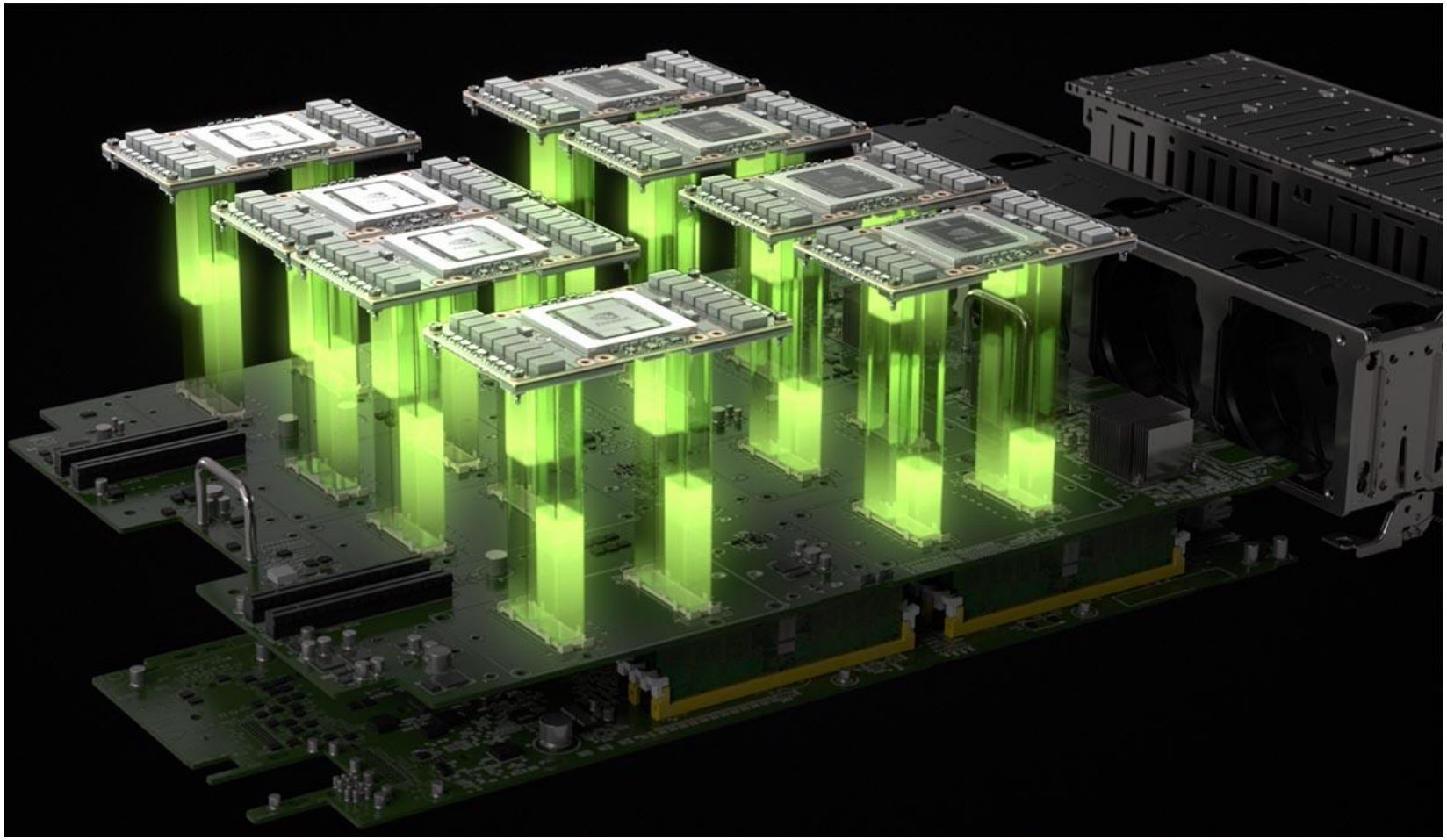
EXAFLOP SISTEMAS

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)

EXAFLOP SISTEMAS

Motivação

- Programar CUDA é algo complexo
- No mínimo é necessário
  - Escrever código para ser executado no host
  - Escrever kernel
  - Mover dados do host para o target
  - Invocar kernel
  - Mover dados do target para o host

Mas queremos que o código rode rápido…

- Escrever código para ser executado no host
- Escrever kernel
- Alocar dados no target
- Mover dados do host para o target
- Invocar kernel de forma assíncrona
- Calcular os limites, índices, distribuição dos multiprocessadores
- Desenrolar laços, abrir atribuições
- Sincronizar no final
- Esperar o kernel terminar
- Mover dados do target para o host, de preferência de forma assíncrona
- Esperar a volta dos dados terminar
- Desalocar variáveis no target
- Terminar computação no target
- Resto…

Motivação

- Algo simples

```
! Redução do array b
a=0.0
do i = 1,n
    a = a + b(i)
end do
```

- Pode se tornar

The reduction code in optimized CUDA

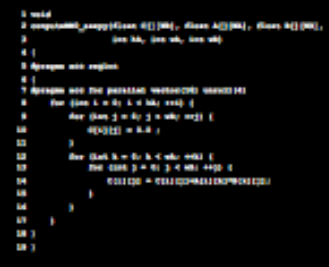
```
template<class T>
struct SharedMemory
{
  __device__ inline operator      T*()
  {
    extern __shared__ int __smem[];
    return (T*)__smem;
  }

  __device__ inline operator const T*() const
  {
    extern __shared__ int __smem[];
    return (T*)__smem;
  }
};

template <class T, unsigned int blockSize, bool nIsPow2>
__global__ void
reduce6(T *g_idata, T *g_odata, unsigned int n)
{
  T *sdata = SharedMemory<T>();

  unsigned int tid = threadIdx.x;
  unsigned int i = blockIdx.x*blockSize*2 + threadIdx.x;
  unsigned int gridSize = blockSize*2*gridDim.x;

  T mySum = 0;
  while (i < n)
  {
    mySum += g_idata[i];
    if (nIsPow2 || i + blockSize < n)
      mySum += g_idata[i+blockSize];
    i += gridSize;
  }
sdata[tid] = mySum;
  __syncthreads();

  if (blockSize >= 512) { if (tid < 256) { sdata[tid] = mySum = mySum
+ sdata[tid + 256]; } __syncthreads(); }
  if (blockSize >= 256) { if (tid < 128) { sdata[tid] = mySum = mySum
+ sdata[tid + 128]; } __syncthreads(); }
  if (blockSize >= 128) { if (tid <  64) { sdata[tid] = mySum = mySum
+ sdata[tid +  64]; } __syncthreads(); }
```

```
  if (tid < 32)
  {
    volatile T* smem = sdata;
    if (blockSize >=  64) { smem[tid] = mySum = mySum + smem[tid + 32];  }
    if (blockSize >=  32) { smem[tid] = mySum = mySum + smem[tid + 16];  }
    if (blockSize >=  16) { smem[tid] = mySum = mySum + smem[tid + 8];  }
    if (blockSize >=   8) { smem[tid] = mySum = mySum + smem[tid + 4];  }
    if (blockSize >=   4) { smem[tid] = mySum = mySum + smem[tid + 2];  }
    if (blockSize >=   2) { smem[tid] = mySum = mySum + smem[tid + 1];  }
  }

  if (tid == 0)
    g_odata[blockIdx.x] = sdata[0];
}
extern "C" void reduce6_cuda_(int *n, int *a, int *b)
{
  int *b_d;
  const int b_size = *n;

  cudaMalloc((void **) &b_d , sizeof(int)*b_size);
  cudaMemcpy(b_d, b, sizeof(int)*b_size, cudaMemcpyHostToDevice);

  dim3 dimBlock(128, 1, 1);
  dim3 dimGrid(128, 1, 1);
  dim3 small_dimGrid(1, 1, 1);
  int smemSize = 128 * sizeof(int);
  int *buffer_d;
  int small_buffer[4],*small_buffer_d;

  cudaMalloc((void **) &buffer_d , sizeof(int)*128);
  cudaMalloc((void **) &small_buffer_d , sizeof(int));
  reduce6<int,128,false><<< dimGrid, dimBlock, smemSize >>>(b_d,buffer_d, b_size);
  reduce6<int,128,false><<< small_dimGrid, dimBlock, smemSize
>>>(buffer_d, small_buffer_d,128);
  cudaMemcpy(small_buffer, small_buffer_d, sizeof(int),
cudaMemcpyDeviceToHost);

  *a = *small_buffer;

  cudaFree(buffer_d);
  cudaFree(small_buffer_d);
  cudaFree(b_d);
}
```

Fonte: GTC2012 Autor: Dr. James C. Beyer, CRAY

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

Matrix Multiply Source Code Size Comparison:



Directives

CUDA C

OpenCL

E a portabilidade?

- Existe? Bom, entre placas NVIDIA (com mesmo “compute capability”) é capaz que sim
- Um código CUDA roda em um equipamento sem CPU?
  - Roda! É só “emular” a GPU
    - Acredito ficar mais lento que um código feito para CPU
- Tenho um Xeon Phi, como fica meu código em CUDA?

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

O OpenACC

OpenACC
Directives for Accelerators

- 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

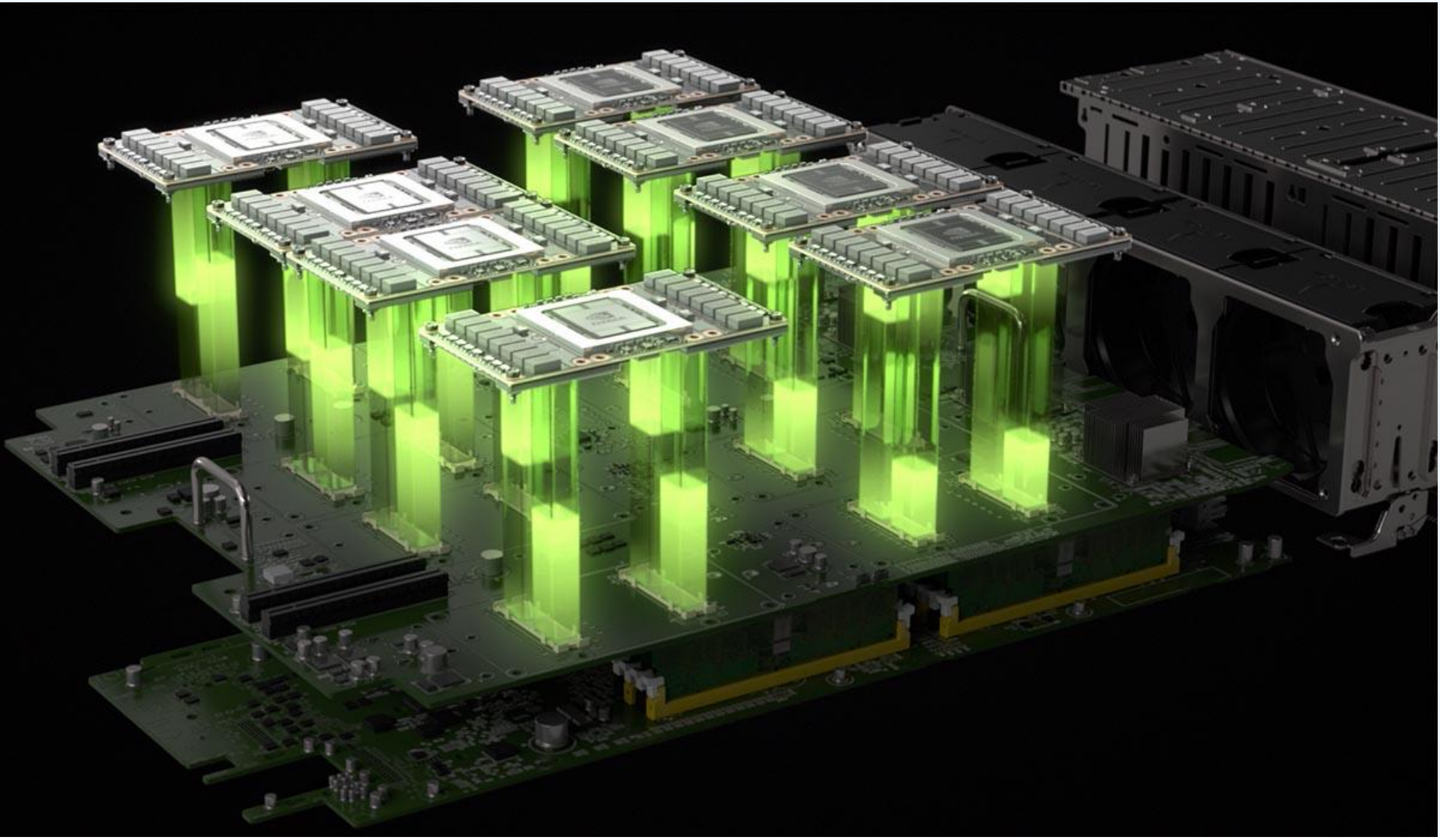
- Poderoso: abstração do paralelismo por parte do compilador permite otimização

O OpenACC

OpenACC
Directives for Accelerators

- 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

NVIDIA Architecture



EXAFLOP SISTEMAS

O OpenACC

OpenACC
Directives for Accelerators

- 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

EXAFLOP SISTEMAS

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;
}
```

EXAFLOP SISTEMAS

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;
}
```

EXAFLOP SISTEMAS

Exemplo: calculando PI com OpenACC

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

	#pragma acc   parallel loop 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;
}
```

EXAFLOP SISTEMAS

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 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;
}
```

EXAFLOP SISTEMAS

Comparação do exemplo #0

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

Outro exemplo!

```
while ( error > tol && iter < iter_max ) {
  error=0.0;

  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]);
    }
  }

  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
  iter++;
}
```

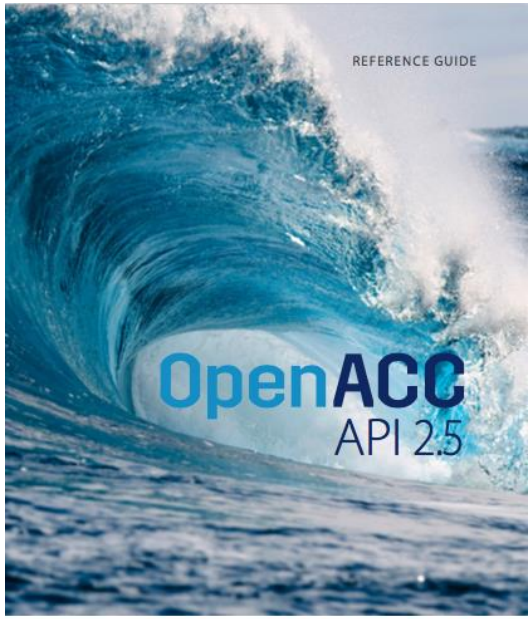
EXAFLOP SISTEMAS

Outro exemplo!

```
while ( error > tol && iter < iter_max ) {
  error=0.0;
#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 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++;
}
```

EXAFLOP SISTEMAS

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]...]*

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

- Quick Reference Guide encontrado em:
- http://www.openacc.org/sites/default/files/OpenACC_2.5_ref_guide_update.pdf

- www.openacc.org
  - NÃO DEIXEM DE VISITAR!!!!!!!!!

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...)

* Novidades da versão 2.0

17,75 segundos

```
while ( error > tol && iter < iter_max ) {
  error=0.0;
#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 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++;
}
```

EXAFLOP SISTEMAS

2,9 segundos

```
#pragma acc data copy(A), create(Anew)
while ( error > tol && iter < iter_max ) {
  error=0.0;
#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 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++;
}
```

EXAFLOP SISTEMAS

COMPILADORES

ANY GOOD SOFTWARE ENGINEER WILL TELL YOU THAT A COMPILER AND AN INTERPRETER ARE INTERCHANGEABLE.

TIM BERNERS LEE

C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg.

— Bjarne Stroustrup —

AZ QUOTES

Java is like a variant of the game of Tetris in which none of the pieces can fill gaps created by the other pieces, so all you can do is pile them up endlessly

Steve Yegge

PICTURE QUOTES.com

You can either have software quality or you can have pointer arithmetic, but you cannot have both at the same time.

— Bertrand Meyer —

AZ QUOTES

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.pgroup.com

Novidade boa

- PGI agora é ”FREE” para a comunidade
  - É uma ”licença sem custo”
- http://www.pgroup.com/products/community.htm

EXAFLOP SISTEMAS

- Pertence … (CLE)
- (foi!) res … ar o padrão C
  - O com … padrão
  - Muito … egiões acelera…
- Um com … ray
  - Ótimo a… , debug, trabalho…
- Quem ter … m

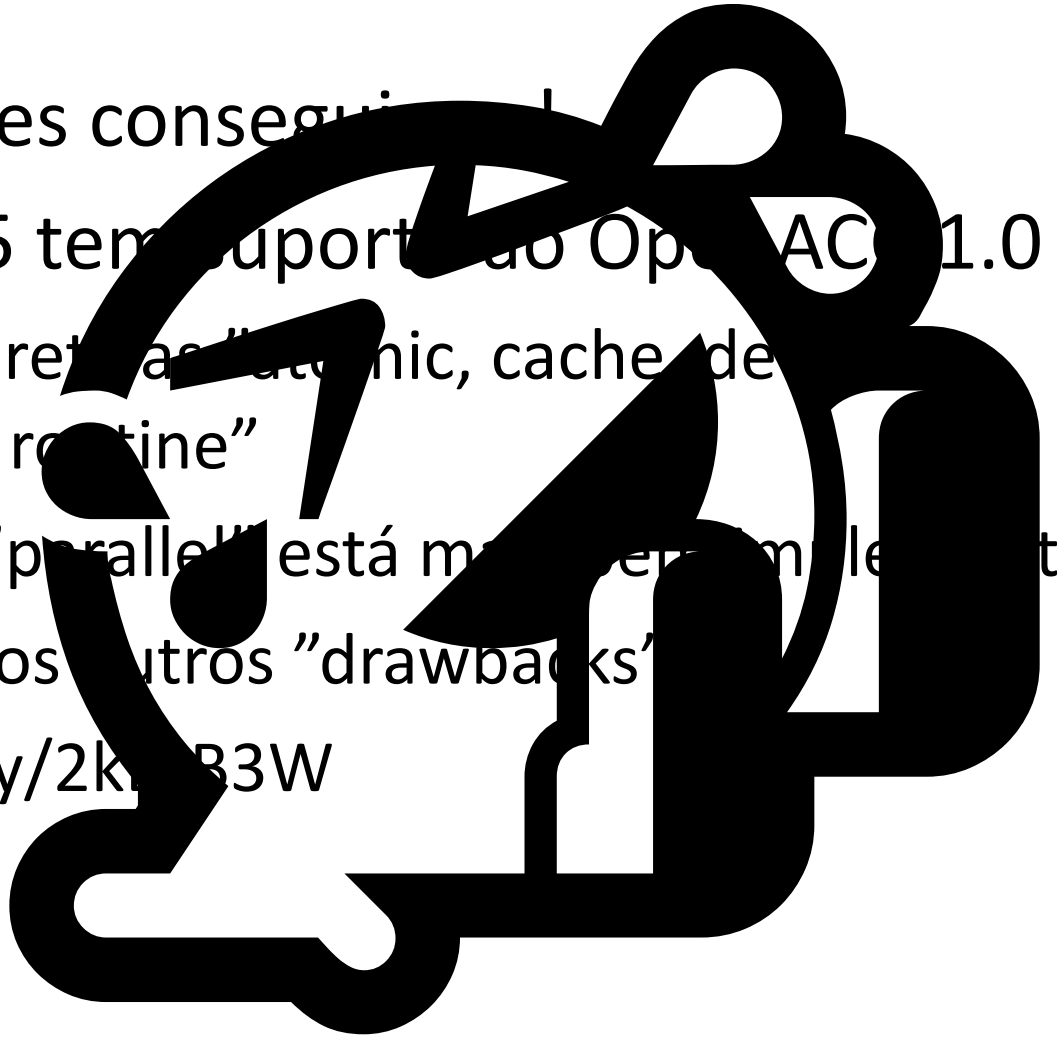
EXAFLOP SISTEMAS

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)

GCC+OpenACC

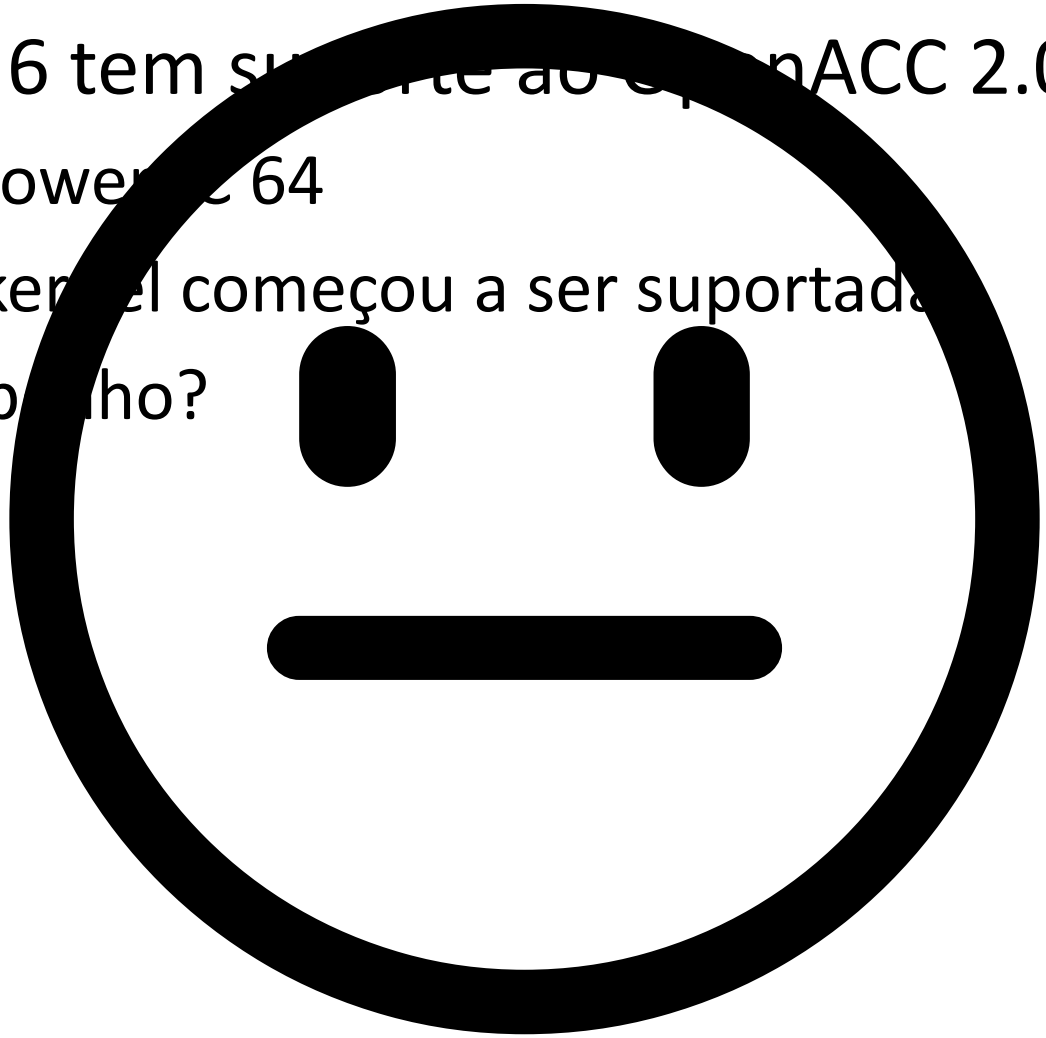
- Adivinha: eles conseguiram [illegible]
- Status: gcc 5 tem suporte [illegible] ao OpenACC 1.0
  - Não tem diretivas "atomic, cache, declare, host_data, routine"
  - A diretiva "parallel" está [illegible] implementada
  - Tem diversos outros "drawbacks"
  - http://bit.ly/2k[illegible]3W



EXAFLOP SISTEMAS

GCC+OpenACC

- Status: GCC 6 tem suporte ao OpenACC 2.0a
  - x86_64 e PowerPC 64
  - A diretiva kernel começou a ser suportada
  - E o desempenho?



Outros compiladores (2017)

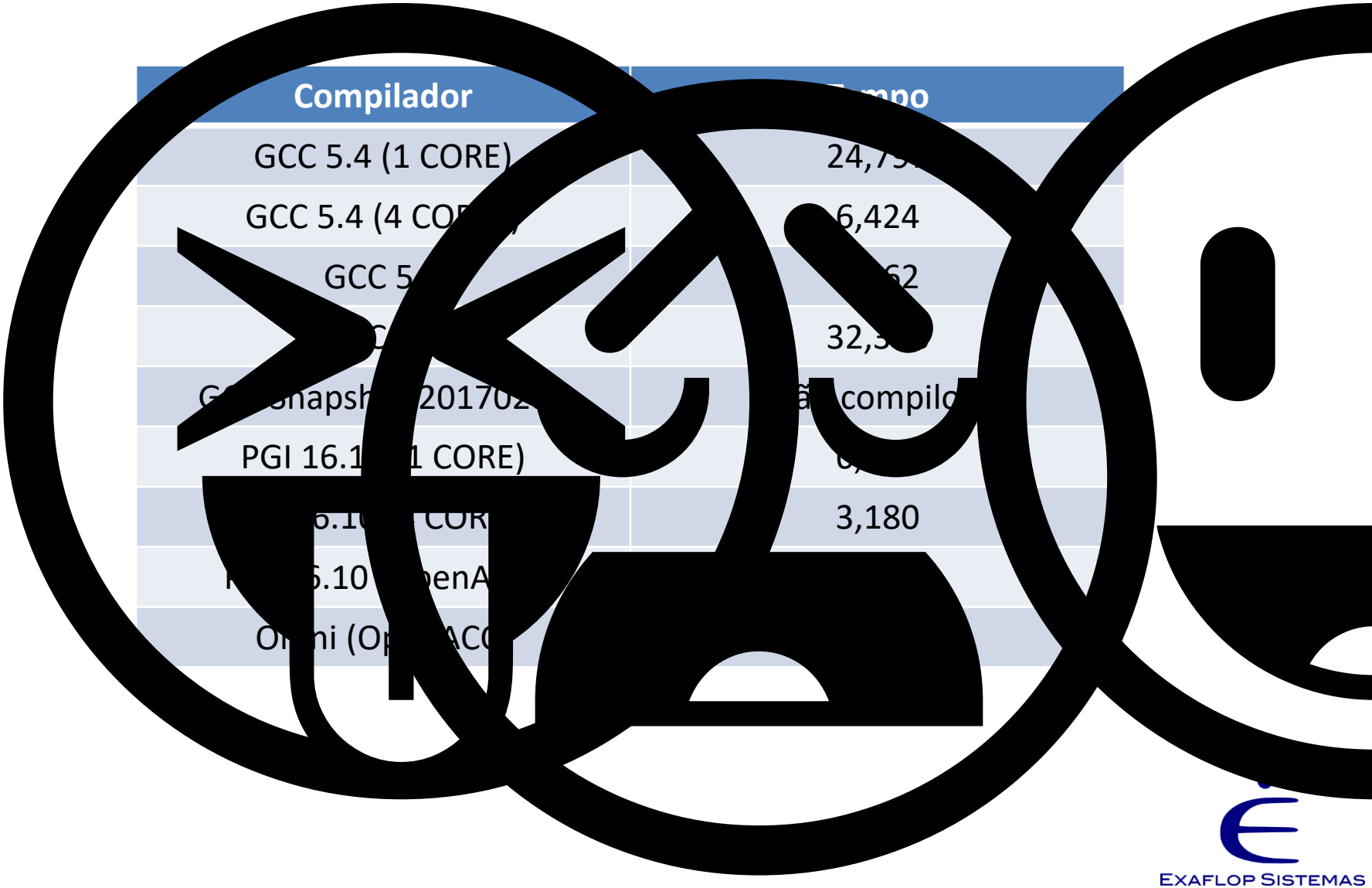
- accULL: Universidad de La Laguna/EPCC
  - Funciona, compila códigos simples, é um “source to source”, convertendo trecho em OpenACC para CUDA, usa Python no meio do caminho…
- Omni: RIKEN/University of Tsukuba (Japão)
  - Melhorou bastante com relação a 2015
  - Teste com o pi: 0,787 segundos (excelente!)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

Compilador	Tempo
GCC 5.4 (1 CORE)	24,75
GCC 5.4 (4 CO	6,424
GCC 5	62
C	32,5
G Snapsh 201702	compilo
PGI 16.1 1 CORE)	6,
6.1 COR	3,180
6.10 penA	
Omni (Op ACC	



EXAFLOP SISTEMAS

Fontes importantes

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

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, 13, 14, 15, 16... da NVIDIA, site www.openacc.org

Obrigado!!!

www.exaflop.com.br

pedro.lopes@exaflop.com.br

EXAFLOP SISTEMAS

ATENÇÃO!!!!

- Continue e divirta-se...

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: AGORA PODE! OpenACC 2.0!
- 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 asíncronas
    - 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!!!

Instalação do PGI (passo-a-passo)

- Ir no site pgroup.com
- 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/linux86-64/2016/bin:$PATH
- export LD_LIBRARY_PATH=

/home/usuario/pgi/linux86-64/2016/lib:$LD_LIBRARY_PATH

- Testar (com os exemplos deste curso...)

Ambiente

- Tenh um computador com placa Nvidia e que esteja funcionando
  - Teste com o comando nvidia-smi
- Obtenha o pacote
- Descompacte-o

Entendendo o pacote (arquivos principais)

- ERAD2017-OpenACC.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 — AXPY para acelerar
- 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

- Utilitário pgaccelinfo (lista aceleradores existentes)
  - Veja também pgcpuid
- Variável de ambiente ACC_DEVICE_NUM controla qual acelerador será usado por padrão
  - Se existir somente um acelerador não é necessária esta variável
- ACC_NOTIFY irá notificar quando executar uma região acelerada
  - Ótimo para debug

Exercício #1: **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 => Speed-up
p => número de processadores
T => 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

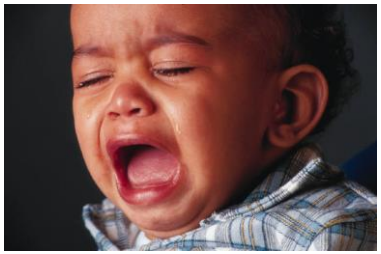
```
#include <stdio.h>
#include <omp.h>
#define N 1000000000
int main(void) {
        double pi = 0.0f; long i;
        printf("Numero de processadores = %d\n", omp_get_max_threads());
        #pragma omp parallel   for reduction(+: pi)   /* OpenMP */
        #pragma acc   parallel loop reduction(+: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

EXAFLOP SISTEMAS

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



EXAFLOP SISTEMAS

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

```
time   ./e1-pi-omp
```

- Dica: com a chave “ –mp ” para controlar o número de processadores use a variável OMP_NUM_THREADS

```
time  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

```
time  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*

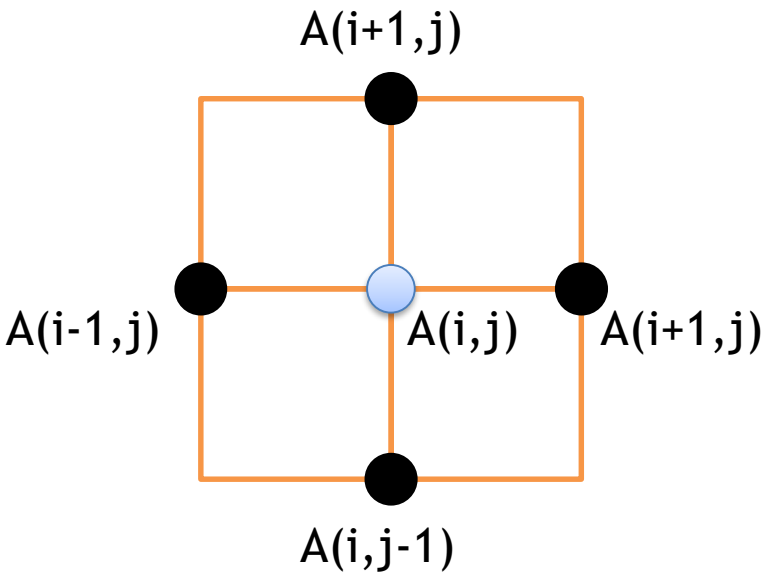
Exercício #2: **e2-laplace2d.c**

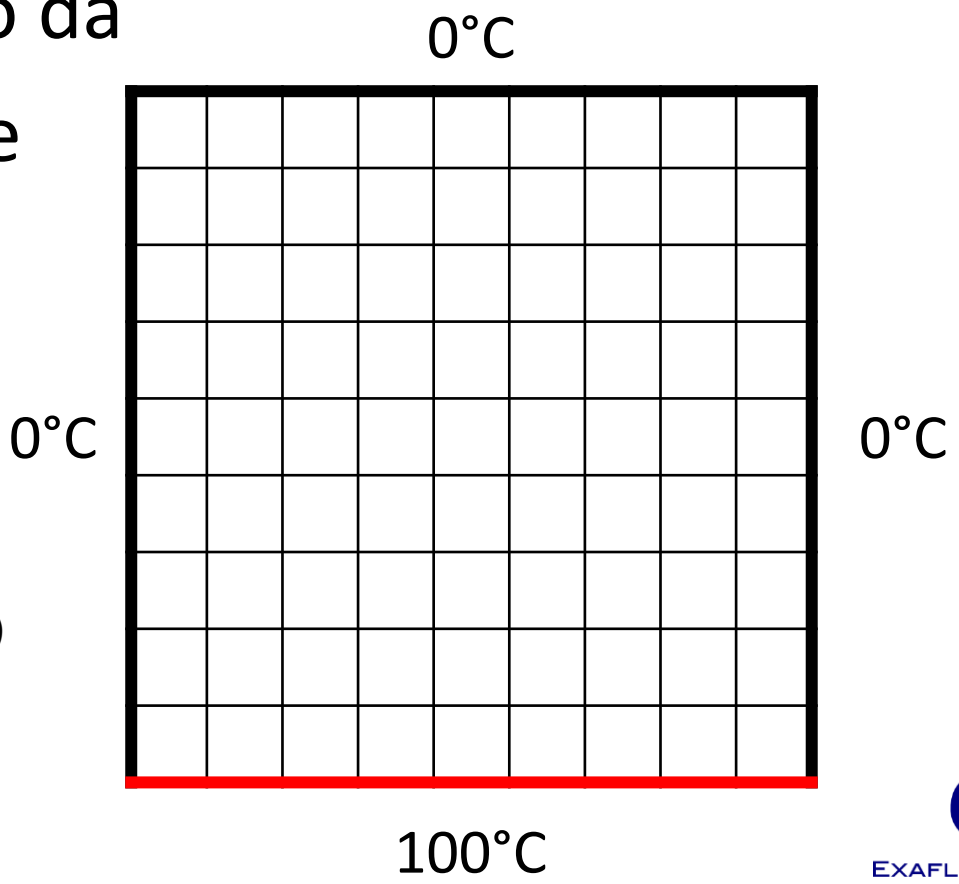
- Objetivo é acelerar um programa mais complexo, com dois laços e um “while”
- Ele já está paralelizado com OpenMP
- Tarefa: encontrar a diretiva e implementar

O problema

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

Equação de Laplace





EXAFLOP SISTEMAS

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

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++;
}
```

EXAFLOP SISTEMAS

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!

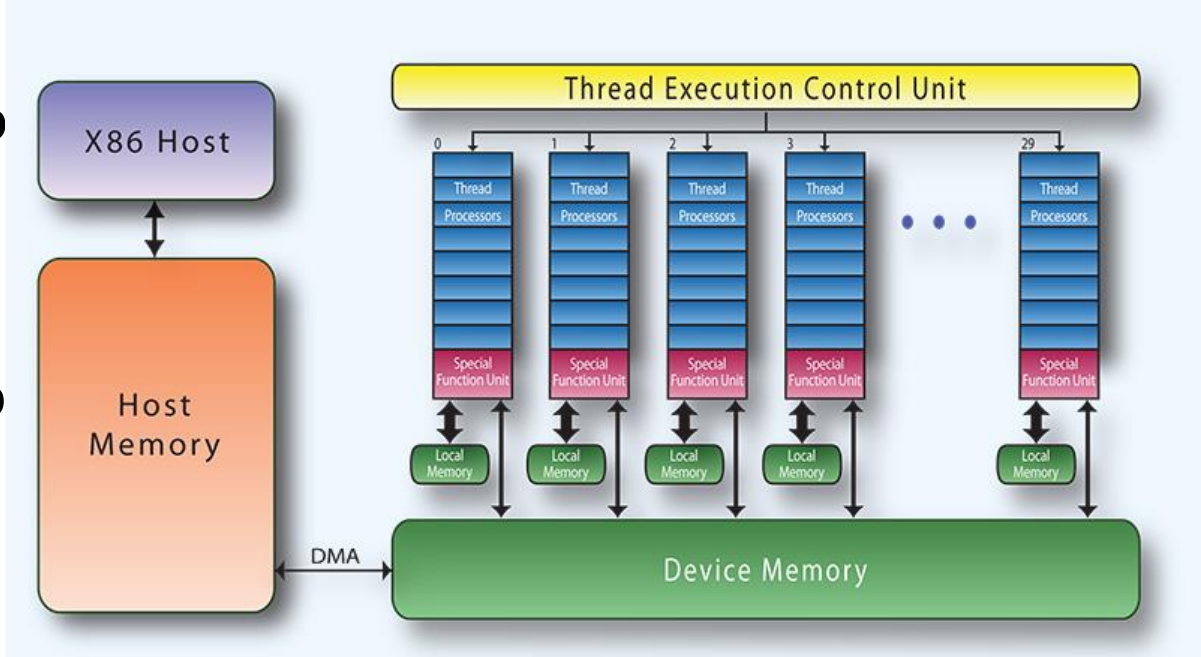
- No host (CPU): 3,81s
- No target(GPU): 17,75s
  - Duas ordens de grandeza a mais de cores!

- 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.



EXAFLOP SISTEMAS

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 algorítmo” 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

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++;
}
```

EXAFLOP SISTEMAS

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)”