Getting Started with Directive-based Acceleration: OpenACC

Ahmad Lashgar
Member of High-Performance Computing Research Laboratory,
School of Computer Science
Institute for Research in Fundamental Sciences (IPM)

October 31, 2012

Introduction

- Heterogeneous systems are energy-efficient design for cluster systems


[Tianhe-1A]


[Nebulae]

[Jaguar Cray XK6]

Heterogeneous System

- Multiple CPUs and Multiple accelerators
  - Each have dedicated DRAM
- CPUs for latency-sensitive workloads
- Accelerators for throughput-intensive workloads

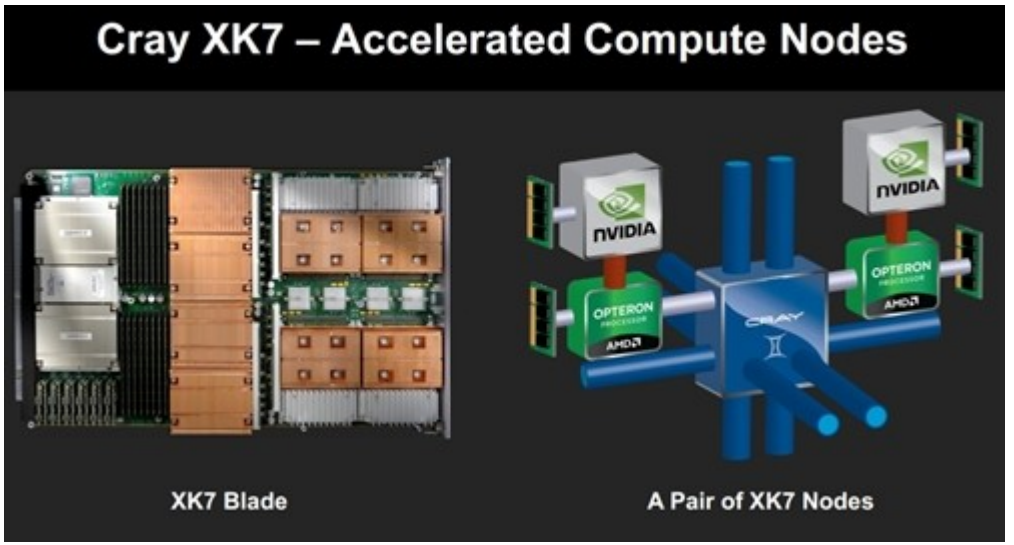
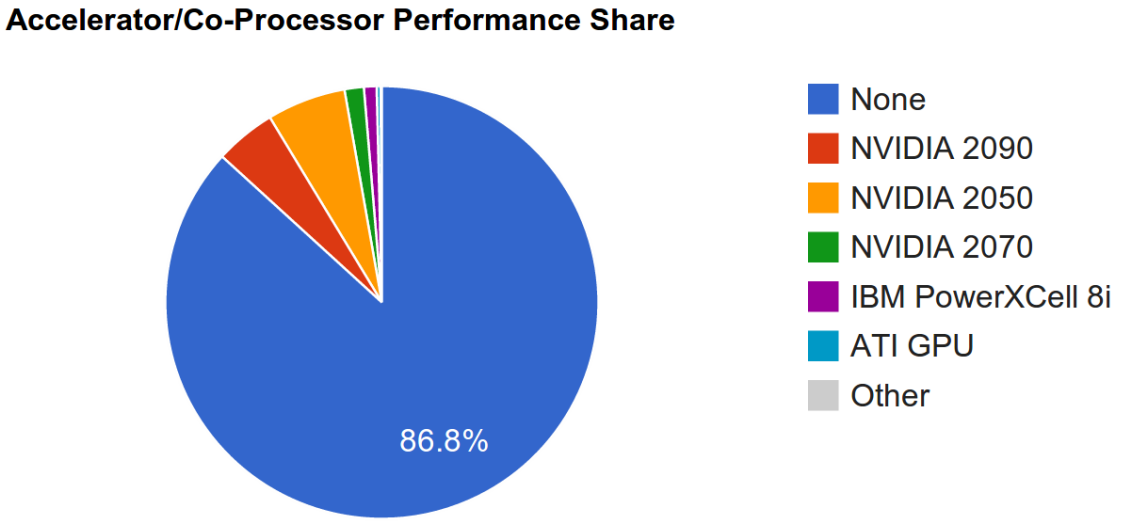


Image Source: NVIDIA

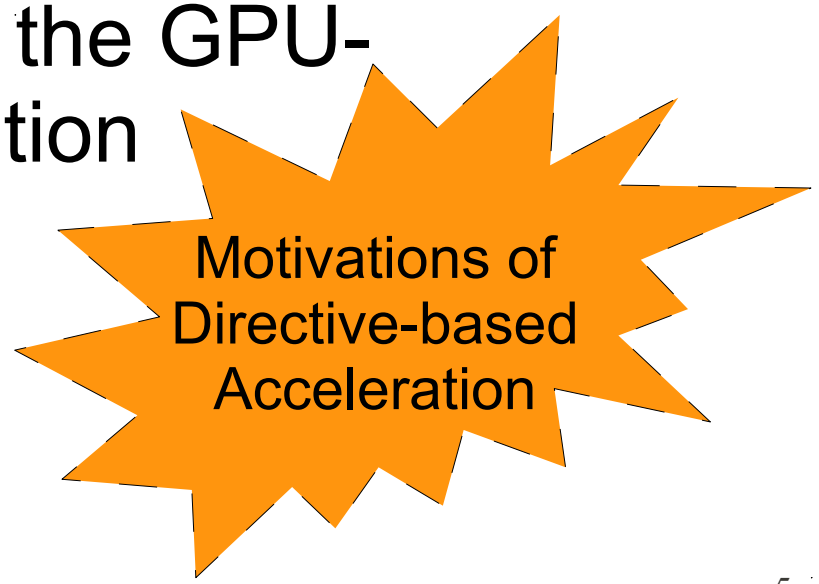
Accelerators

- Conventional accelerators are NVIDIA/ATI GPUs
- Accelerator deployment shared in HPC systems according to Top500



Well-Known Programming Models

- Conventional programming models:
  - CUDA
  - OpenCL
- Not an easy task
- Costly initial effort to check the GPU-friendliness of CPU application



Outline

- Introduction
- Directive-based Acceleration
- Immersion in OpenACC
- OpenACC Standard in C
  - Directives
  - Library Routines
- Case Study: n-Body
- Conclusion

Directive-Based Acceleration

- Why not OpenMP accelerator extension [3]
  - Expose architecture to let tunning/optimization
    - No host-device concurrency
    - No vocabulary on separate memory space
- CAPS HMPP (2007)
  - PathScale ENZO Compiler Suite
  - CAPS
- OpenACC (2011)
  - PGI Compiler
  - CAPS
  - ? INRIA StartPU GCC plug-in [6]
  - ? accULL [7] YACF [8]

Immersion in OpenACC

- We used PGI Accelerator Compiler
  - 15-day trial available
- Matrix Add
  - Code modification
  - Performance profiling and tunning
  - Resulting code

Matrix Add

```
#include <malloc.h>
#include <time.h>

#define SIZE 1000
int main()
{
        int i,j;

        float **a, **b, **c, **seq;
        a=(float**)malloc(SIZE*sizeof(float*));
        b=(float**)malloc(SIZE*sizeof(float*));
        c=(float**)malloc(SIZE*sizeof(float*));
        seq=(float**)malloc(SIZE*sizeof(float*));
        for(i=0; i<SIZE; i++){
                a[i]=(float*)malloc(SIZE*sizeof(float));
                b[i]=(float*)malloc(SIZE*sizeof(float));
                c[i]=(float*)malloc(SIZE*sizeof(float));
                seq[i]=(float*)malloc(SIZE*sizeof(float));
        }
```

Matrix Add (2)

```
// Initialize matrices.
for (i = 0; i < SIZE; ++i) {
        for (j = 0; j < SIZE; ++j) {
                a[i][j] = (float)i + j;
                b[i][j] = (float)i - j;
                c[i][j] = 0.0f;
        }
}

unsigned long long int tic, toc;
// Compute matrix Add
printf("Calculation on GPU ... ");
tic = clock();
#pragma acc kernels copyin(a[0:SIZE][0:SIZE],b[0:SIZE][0:SIZE]) copy(c[0:SIZE][0:SIZE])
{
for (i = 0; i < SIZE; ++i) {
        for (j = 0; j < SIZE; ++j) {
                c[i][j] = a[i][j] + b[i][j];
        }
}
}
toc = clock();
printf(" %6.4f ms\n",(toc-tic)/(float)1000);
```

Matrix Add (3)

```
    // Perform the add
    printf("Calculation on CPU ... ");
    tic = clock();
    for (i = 0; i < SIZE; ++i) {
        for (j = 0; j < SIZE; ++j) {
            seq[i][j] = a[i][j] + b[i][j];
            if(c[i][j] != seq[i][j]) {
                printf("Error %d %d\n", i,j);
                exit(1);
            }
        }
    }
    toc = clock();
    printf(" %6.4f ms\n",(toc-tic)/(float)1000);

    printf("OpenACC vector add test was successful!\n");

    return 0;
}
```

Compiler Flags

- Generate acc regions for two binaries
  - -ta=nvidia,host
- Enable acc regions on accelerator only:
  - -ta=nvidia
- Watch through the generated GPU or PTX code
  - -ta=nvidia,keepptx,keepgpu
- Change target hardware
  - -ta=nvidia,cc13
  - -ta=nvidia,cc20

Compiling

```
$ pgcc -acc -fast -Minfo vectorAdd.c -o vectorAdd
main:
     17, Loop not vectorized/parallelized: contains call
     27, Loop not vectorized: data dependency
         Loop unrolled 8 times
     38, Generating present_or_copyin(b[0:1000][0:1000])
         Generating present_or_copyin(a[0:1000][0:1000])
         Generating present_or_copy(c[0:1000][0:1000])
         Generating compute capability 1.0 binary
         Generating compute capability 2.0 binary
     40, Complex loop carried dependence of '*(*(b))' prevents parallelization
         Complex loop carried dependence of '*(*(a))' prevents parallelization
         Complex loop carried dependence of '*(*(c))' prevents parallelization
         Accelerator scalar kernel generated
         CC 1.0 : 11 registers; 40 shared, 4 constant, 0 local memory bytes
         CC 2.0 : 22 registers; 0 shared, 56 constant, 0 local memory bytes
     41, Complex loop carried dependence of '*(*(b))' prevents parallelization
         Complex loop carried dependence of '*(*(a))' prevents parallelization
         Complex loop carried dependence of '*(*(c))' prevents parallelization
     56, Loop not vectorized/parallelized: contains call
```

Results

```
Calculation on GPU ...  2570.0000 ms
Calculation on CPU ...  0.0000 ms
OpenACC vector add test was successful!
```

- Poor performance!
- Lets Start Profiling
  - PGI_ACC_TIME
  - ACC_NOTIFY

Profiling: export PGI_ACC_TIME=1

```
$ ./vectorAdd
Calculation on GPU ...  2520.0000 ms
Calculation on CPU ...  10.0000 ms
OpenACC vector add test was successful!

Accelerator Kernel Timing data
/proj/pgrel/extract/x86/2012/rte/accel/hammer/lib-linux86-64/../src-nv/nvfill.c
  __pgi_cu_fill
    26: region entered 3 times
        time(us): total=5,205
                  kernels=34
        27: kernel launched 3 times
            grid: [8]  block: [128]
            time(us): total=34 max=16 min=9 avg=11
/share/users/alashgar/openacc/vectorAdd.c
  main
    38: region entered 1 time
        time(us): total=2,711,316 init=1,466,389 region=1,244,927
                  kernels=1,208,930 data=35,441
        w/o init: total=1,244,927 max=1,244,927 min=1,244,927 avg=1,244,927
        40: kernel launched 1 times
            grid: [1]  block: [1]
            time(us): total=1,208,930 max=1,208,930 min=1,208,930 avg=1,208,930
```

Profiling: export ACC_NOTIFY=1

```
$ ./vectorAdd
      launch kernel  file=/proj/pgrel/extract/x86/2012/rte/accel/hammer/lib-linux86-64/../src-nv/nvfill.c function=__pgi_cu_fill line=27 device=0 grid=8 block=128 queue=0
      launch kernel  file=/proj/pgrel/extract/x86/2012/rte/accel/hammer/lib-linux86-64/../src-nv/nvfill.c function=__pgi_cu_fill line=27 device=0 grid=8 block=128 queue=0
      launch kernel  file=/proj/pgrel/extract/x86/2012/rte/accel/hammer/lib-linux86-64/../src-nv/nvfill.c function=__pgi_cu_fill line=27 device=0 grid=8 block=128 queue=0
      launch kernel  file=/proj/pgrel/extract/x86/2012/rte/accel/hammer/lib-linux86-64/../src-nv/nvfill.c function=__pgi_cu_fill line=40 device=0 grid=1 block=1 queue=0
Calculation on GPU ...  2470.0000 ms
Calculation on CPU ...  0.0000 ms
OpenACC vector add test was successful!
```

Understanding Compiler Report

```
$ pgcc -acc -fast -Minfo vectorAdd.c -o vectorAdd
NOTE: your trial license will expire in 13 days, 11.5 hours.
main:
     17, Loop not vectorized/parallelized: contains call
     27, Loop not vectorized: data dependency
         Loop unrolled 8 times
     38, Generating present_or_copyin(b[0:1000][0:1000])
         Generating present_or_copyin(a[0:1000][0:1000])
         Generating present_or_copy(c[0:1000][0:1000])
         Generating compute capability 1.0 binary
         Generating compute capability 2.0 binary
     40, Complex loop carried dependence of '*(*(b))' prevents parallelization
         Complex loop carried dependence of '*(*(a))' prevents parallelization
         Complex loop carried dependence of '*(*(c))' prevents parallelization
         Accelerator scalar kernel generated
         CC 1.0 : 11 registers; 40 shared, 4 constant, 0 local memory bytes
         CC 2.0 : 22 registers; 0 shared, 56 constant, 0 local memory bytes
     41, Complex loop carried dependence of '*(*(b))' prevents parallelization
         Complex loop carried dependence of '*(*(a))' prevents parallelization
         Complex loop carried dependence of '*(*(c))' prevents parallelization
     56, Loop not vectorized/parallelized: contains call
```

Modifying the matrix add acc

```
#pragma acc data copyin(a[0:SIZE][0:SIZE],b[0:SIZE][0:SIZE]) copy(c[0:SIZE][0:SIZE])
{
      # pragma acc region
      {
            #pragma acc loop independent vector(16)
            for (i = 0; i < SIZE; ++i) {
                  #pragma acc loop independent vector(16)
                  for (j = 0; j < SIZE; ++j) {
                        c[i][j] = a[i][j] + b[i][j];
                  }
            }
      }
}
```

Modifying the matrix add acc (2)

- First time kernel launch is costly, evaluate it over iterations

```
for(int k=0; k<3; k++){
        printf("Calculation on GPU ... ");
        tic = clock();

        #pragma acc data pcopyin(a[0:SIZE][0:SIZE],b[0:SIZE][0:SIZE]) pcopy(c[0:SIZE][0:SIZE])
        {
                # pragma acc region
                {
                        #pragma acc loop independent vector(16)
                        for (i = 0; i < SIZE; ++i) {
                                #pragma acc loop independent vector(16)
                                for (j = 0; j < SIZE; ++j) {
                                        c[i][j] = a[i][j] + b[i][j];
                                }
                        }
                }
        }
        toc = clock();
        printf(" %6.4f ms\n",(toc-tic)/(float)1000);
}
```

Results

launch kernel file=/share/users/alashgar/openacc/vectorAdd3.c function=main line=47 device=0 grid=63x63 block=16x16 queue=0
Calculation on GPU ... 1300.0001 ms
launch kernel file=/share/users/alashgar/openacc/vectorAdd3.c function=main line=47 device=0 grid=63x63 block=16x16 queue=0
Calculation on GPU ... 30.0000 ms
launch kernel file=/share/users/alashgar/openacc/vectorAdd3.c function=main line=47 device=0 grid=63x63 block=16x16 queue=0
Calculation on GPU ... 40.0000 ms
Calculation on CPU ... 0.0000 ms
OpenACC vector add test was successful!

Tips

- From PGI compiler manual:
  - “When your program reaches its first accelerator region, there may be a 0.5 to 1.5 second pause to warm up the GPU from a power-off condition.”
- Tip 1:
  - Use acc_init in early lines to prevent cold initial delay:

```
#include <accel.h>
#include <accelmath.h>

acc_init( acc_device_nvidia );
```

Results (2)

- Speedup shows up for larger the parallel operations:

sin(a[i][j]) + cos(b[i][j]) + cos(a[i][j]*b[i][j])

```
$ ./vectorAdd4
launch kernel  file=/share/users/alashgar/openacc/vectorAdd4.c function=main
line=51 device=0 grid=63x63 block=16x16 queue=0
Calculation on GPU ...  40.0000 ms
launch kernel  file=/share/users/alashgar/openacc/vectorAdd4.c function=main
line=51 device=0 grid=63x63 block=16x16 queue=0
Calculation on GPU ...  30.0000 ms
launch kernel  file=/share/users/alashgar/openacc/vectorAdd4.c function=main
line=51 device=0 grid=63x63 block=16x16 queue=0
Calculation on GPU ...  40.0000 ms
Calculation on CPU ...  70.0000 ms
```

Outline

- Introduction
- Directive-based Acceleration
- Immersion in OpenACC
- OpenACC Standard in C
  - Directives
  - Library Routines
- Case Study: n-Body
- Conclusion

OpenACC Standard in C

- Standardized by PGI, NVIDIA, CAPS, and Cray
  - [date]
- Available controls:
  - Directives
  - Runtime Library Routines

Directives

- General

  #pragma acc directive-name [clause...]

- Directive-name
  - parallel
  - kernels
  - loop
  - data

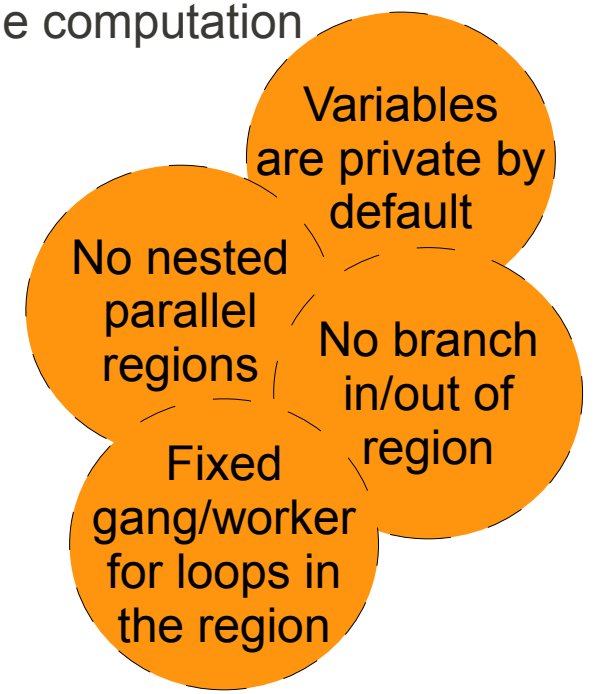
Parallel Construct

#pragma acc parallel [clause]

- Clauses
  - if (condition)
  - async (expression)
  - num_gangs ()
  - num_workers ()
  - vector_length ()
  - private (list)
  - firstprivate (list)
  - reduction (operation:list)

    +, *, min, max, &, |, ^, &&, ||

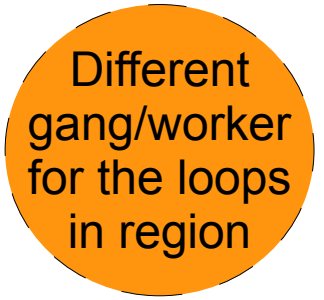
```
#pragma acc parallel
for ( i=0; i<N; i++)
{
        // core of the computation
}
```



Kernels Construct

#pragma acc kernels [clause]

- Clauses
  - if (condition)
  - async (expression)



Loop Construct

#pragma acc loop [clause]

- Clauses
  - seq
  - independent
  - collapse (n-nested-loops)
  - private (list)
  - reduction (operator:list)

Loop Construct (2)

#pragma acc loop [clause]

- Clauses for loop in parallel
  - gang
  - worker
  - vector

Loop Construct (3)

#pragma acc loop [clause]

- Clauses for loop in kernel
  - independent
  - gang [(num_gangs)]
  - worker [(num_workers)]
  - vector [(num_vectors)]

Data Construct

#pragma acc data [clause]

- Clause
  - copy(list) /pcopy(list)
  - copyin(list) /pcopyin(list)
  - copyout(list) /pcopyout(list)
  - create(list) /pcreate(list)

Data Construct (2)

```
for(int k=0; k<10; k++){
        printf("Calculation on GPU ... ");
        tic = clock();

        #pragma acc data pcopyin(a[0:SIZE][0:SIZE],b[0:SIZE][0:SIZE]) pcopy(c[0:SIZE][0:SIZE])
        {
                # pragma acc region
                {
                        #pragma acc loop independent vector(16)
                        for (i = 0; i < SIZE; ++i) {
                                #pragma acc loop independent vector(16)
                                for (j = 0; j < SIZE; ++j) {
                                        c[i][j] = a[i][j] + b[i][j];
                                }
                        }
                }
        }
        toc = clock();
        printf(" %6.4f ms\n",(toc-tic)/(float)1000);
}
```

Runtime Library Routines

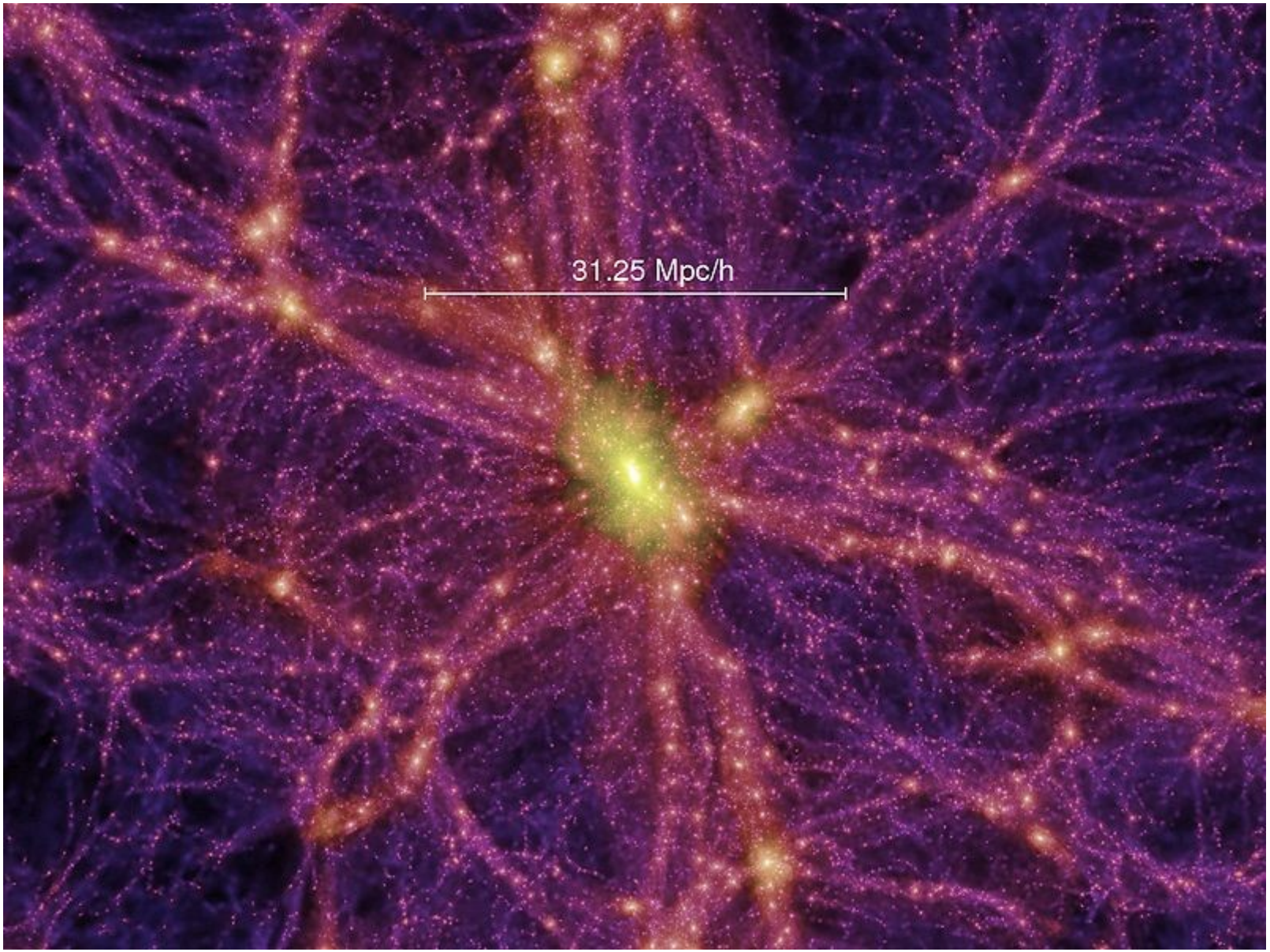
#include “openacc.h”

- acc_get_num_devices (devicetype)
- acc_set_device_type (devicetype)
- acc_get_device_type (devicetype)
- acc_set_device_num (devicenum, devicetype)
- acc_get_device_num (devicetype)
- acc_async_wait (expression)
- acc_async_wait_all ()
- acc_init (devicetype)

Outline

- Introduction
- Directive-based Acceleration
- Immersion in OpenACC
- OpenACC Standard in C
  - Directives
  - Library Routines
- Case Study: n-Body
- Conclusion

Case Study: N-Body



N-Body CPU

- We start with NVIDIA SDK sample

```
_integrateNBodySystem(float deltaTime)
{
    _computeNBodyGravitation();
    for (int i = 0; i < m_numBodies; ++i)
    {
        // couple of operations
    }
}
```

```
_computeNBodyGravitation()
{
    for(int i = 0; i < m_numBodies; ++i)
    {
            for(int j = 0; j < m_numBodies; ++j)
            {
                // couple of operations
            }
    }
}
```

N-Body OpenMP

```
_integrateNBodySystem(float deltaTime)
{
    _computeNBodyGravitation();

    #pragma omp parallel private(...)
    {
        #pragma omp for
        for (i = 0; i < m_numBodies; ++i)
        {
            // couple of operations
        }
    }
}
```

```
_computeNBodyGravitation()
{
    #pragma omp parallel for private(j,...)
    for(i = 0; i < m_numBodies; ++i)
    {
            for(j = 0; j < m_numBodies; ++j)
            {
                // couple of operations
            }
    }
}
```

N-Body CUDA

- Tile-based one-two-one force calculation

```
__global__ void integrateBodies
      ( float4* newPos, float4* newVel, float4* oldPos, float4* oldVel,
        float deltaTime, float damping, int numBodies)
{
    int index = __mul24(blockIdx.x,blockDim.x) + threadIdx.x;
    float4 pos = oldPos[index];

    float3 accel = computeBodyAccel<multithreadBodies>(pos, oldPos, numBodies);
    // couple of operations

    // store new position and velocity
    newPos[index] = pos;
    newVel[index] = vel;
}
```

N-Body OpenACC

```
BodySystemCPU::_integrateNBodySystem(float deltaTime)
{
        _computeNBodyGravitation( … );
        _updatePosVel( … );
}
```

N-Body OpenACC (2)

```
void _computeNBodyGravitation( … )
{
    #pragma acc data copy(m_force[0:m_numBodies*4],m_pos[0][0:m_numBodies*4])
    {
        #pragma acc kernels loop gang(32), worker(256)
        {
            #pragma acc loop independent collapse(2)
            for(i = 0; i < m_numBodies; ++i)
            {
                for(j = 0; j < m_numBodies; ++j)
                {
                    // couple of operations
                }
            }
        }
    }
}
```

N-Body OpenACC (3)

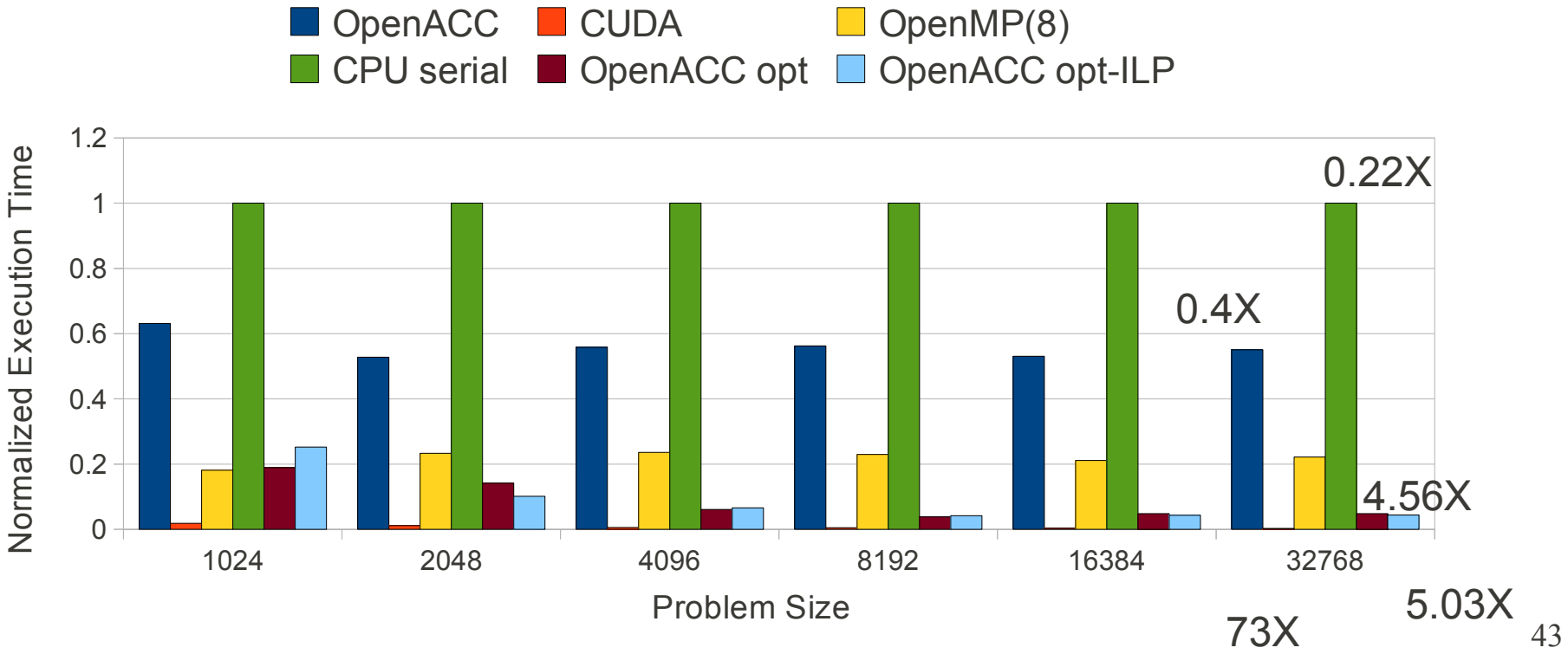
```
void _updatePosVel( … )
{
        #pragma acc data copyin( \
              m_force[0:m_numBodies*4], \
              m_pos[0][0:m_numBodies*4], \
              m_vel[0][0:m_numBodies*4])
        #pragma acc data copyout( \
              m_pos[1][0:m_numBodies*4], \
              m_vel[1][0:m_numBodies*4])
        #pragma acc region
        {
              #pragma acc loop independent
               for (int i = 0; i < m_numBodies; ++i)
               {
                    // couple of operations
               }
      }
}
```

N-Body OpenACC Optimized

- Compile for system GPU
  - -ta=nvidia,cc13
- Remote inline call and do hand optimize
  - Turn private arrays to registers
  - Provide enough ILP by code reordering

Results

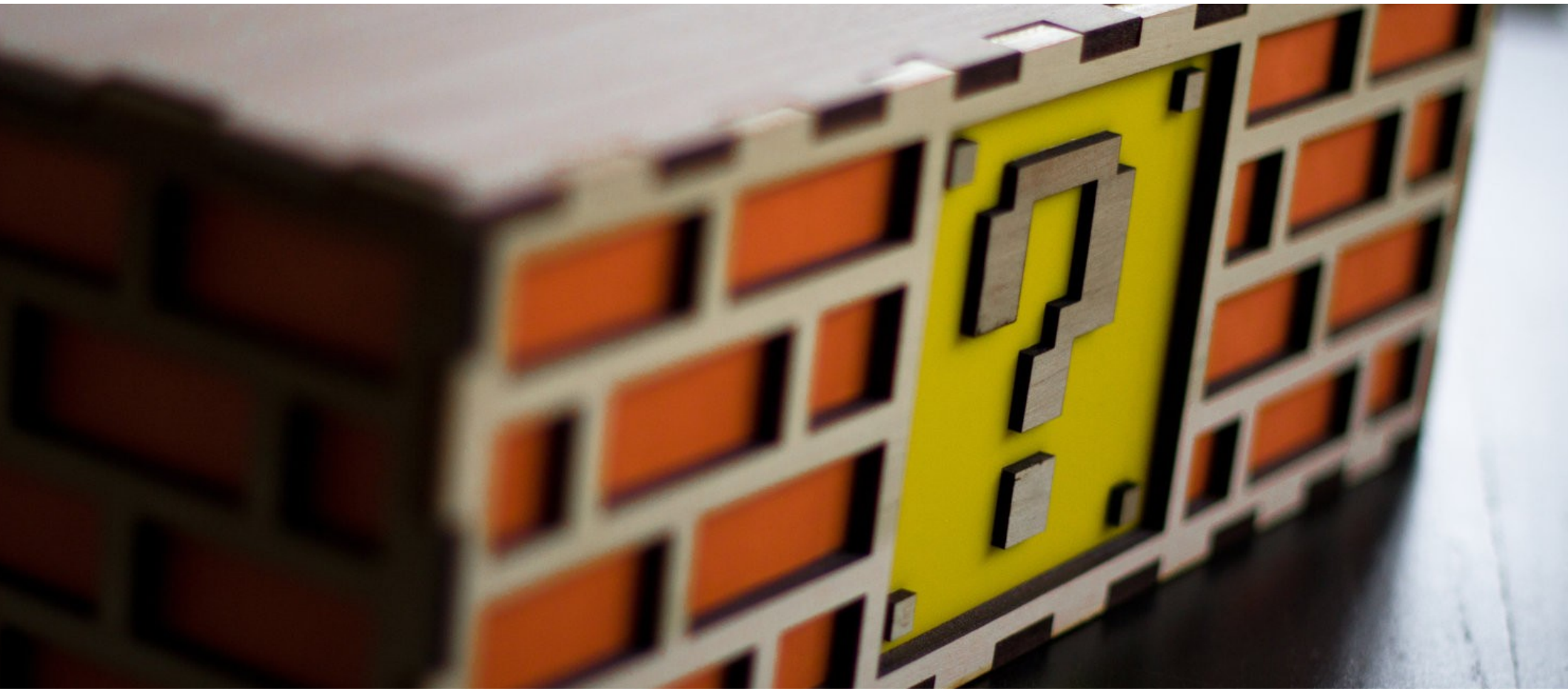
- Normalized to OpenMP(8)
  - Average of 6 runs
  - CPU: Core-i7. GPU: GTX 295



Conclusion

- OpenACC is easy-to-use acceleration-based programming model
- CUDA and OpenACC pointer passing
- Remaining challenges
  - Public support from giant accelerator manufacturers
  - Free compiler

Thank you for attention!



References

[1] http://www.pgroup.com/lit/articles/insider/v1n1a1.htm
[2] http://www.pgroup.com/doc/pgiug.pdf
[3] http://www.pgroup.com/lit/articles/insider/v2n2a5.htm
[4] http://cs.ioc.ee/etaps12/invited/bodin-slides.pdf
[5] http://www.caps-entreprise.com/wp-content/uploads/2012/08/caps-hmpp-gpgpu-Boston-Workshop-Oct-2007.pdf
[6] http://article.gmane.org/gmane.comp.gcc.devel/125551
[7] https://accull.wordpress.com/
[8] https://code.google.com/p/yacf/source/checkout
[TIANHE-1A] http://www.xbitlabs.com/images/other/decades10/nvidia_tianhe.jpg
[Jaguar Cray XK6] http://blog.seattletimes.nwsource.com/brierdudley/jaguar-7.jpg
[Nebulae] https://supercomputers2011.files.wordpress.com/2011/09/nebulae.jpg

Wait Directive

#pragma acc wait

- Causes the host program to wait for completion of asynchronous accelerator activies

Cache Directive

#pragma acc cache (list)

- May be added to top of the loop. The elements and sub-array in the list are cached in the software-managed data cache.