Accelerating GADGET
Early experience for GPU-acceleration using OpenMP 4.0

Thilo Maurer, Lars Klein, IBM Research & Development Böblingen GmbH
Tuesday, 13 October 2015, Jülich Supercomputing Centre, PADC Opening Workshop
Table of contents

GADGET

OpenMP 4.0
Brief introduction of the astrophysics code

What is new with version 4.0 of the OpenMP specification

LLVM C/C++ compiler with OpenMP 4.0 support
Presentation of the clang OpenMP compiler

Insights & Experiences
Gathered about GADGET and offloading with OpenMP

Performance results

Summary
GADGET
Brief introduction of the astrophysics code
What is GADGET?

- Code for cosmological N-body/SPH simulations
- Part of the PRACE Unified European Applications Benchmark Suite
- GADGET uses MPI to run simulations on parallel computers with distributed memory
- Used for „Millenium Simulation“
- Volker Springel at the University of Heidelberg
- Gadget 4: under development
  - OpenMP for in node parallelism
  - CUDA/OpenCL for time-consuming parts
N-body simulation (direct summation)

- Given N bodies in the simulation space at some initial time
  - Each body has position, velocity and mass
  - Mutual interactions by some force, e.g., gravitational forces
  - (The positions of all bodies are pairwise disjunct)

- The simulation approximates the position and velocity of all bodies at given later time
  - Calculate net force (look at each particle) for each particle

- Use leap-frog integration to iterate through time

- Complexity: O(n²)
The Barnes-Hut tree algorithm

- To reduce the computational complexity GADGET uses a hierarchical tree algorithm
- Complexity: $O(n \cdot \log(n))$
- Tree algorithm consumes near 70% of the execution time (for single threaded application)
  - That is the reason why we focused on this kernel of GADGET
- Basic idea of the algorithm:
  - Organize bodies by recursively splitting the simulation box into $2 \times 2 \times 2$ equally sized “octants”
  - A cube is not divided into another 8 cubes if it contains only one body
- Allows for trade-off:
  - Instead of calculating the forces between every body, represent a group of bodies too-distant to the current body as a single body
  - The position of this body is the group's center of mass and the mass is the group's total mass
Example of the tree algorithm
Sub-Division of simulation space

Physical

Logically resulting tree

- **A** internal node
- **B** empty node
- **F** body

Diagram showing the division of the simulation space and the corresponding tree structure.
Tree-walk example

- Let body $A$ be the one for which we do the tree-walk
- First start with the root node, representing all bodies in the simulation space
  - All bodies represented by the root node are not distant enough to body $A$, so look at the children
Tree-walk example

- The first child contains body A
  - Ignore, since A do not need exert force on itself

- The second child is an internal node containing multiple bodies
  - The bodies represented by this node are again *not distant enough* to body A
  - Take a look at the children of this node
Tree-walk example

- The first child contains only body B
  - Calculate the forces between A and B
- The second child contains only body C
  - Calculate the forces between A and C
- The third child is an empty node
- The forth child is an internal node containing multiple bodies
The bodies represented by the fourth child are distant enough to body A
- Approximate bodies D and E through the internal node
- Calculate the forces between A and the internal node
- position-of-node = center-of-mass ; mass-of-node = sum-of-masses
- This is the way calculations are saved
Tree-walk example

- We are done with the first and second child of the root node, but still need to look at children number three and four
- The third child is an empty node
  - Nothing needs to be done
- The fourth child contains only body F
  - Calculate the forces between A and F
# Table of contents

**GADGET**

- Brief introduction of the astrophysics code

**OpenMP 4.0**

- What is new with version 4.0 of the OpenMP specification

**LLVM C/C++ compiler with OpenMP 4.0 support**

- Presentation of the clang OpenMP compiler

**Insights & Experiences**

- Gathered about GADGET and offloading with OpenMP

**Performance results**

**Summary**
OpenMP 4.0
What is new with version 4.0 of the OpenMP specification
OpenMP 4.0

- Specification 4.0 of OpenMP was released in Summer '13
  - Version 4.1 in preparation

- Introduced new features for OpenMP
  - Support for accelerators
  - SIMD constructs to vectorize both serial as well as parallelized loops
  - Thread affinity
  - And some more

- Focus on the accelerator support for this project
Acceleration support – data transfer

- `#pragma omp target data <clause>`
  - Create a device data environment for the extent of the region
  - One of the possible clauses: `map([to:], [from:], [tofrom:] list)`

- `#pragma omp declare target`
  - A declarative directive that specifies that variables and functions are mapped to a device

- `#pragma omp target update <motion-clause>`
  - Makes the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses (`to` or `from`)
Acceleration support – kernel execution

- `#pragma omp target <clause>`
  - Same as target data, but also starts execution on the device

- `#pragma omp teams <clause>`
  - Creates a league of thread teams where the master thread of each team executes the region
  - `num_teams()` and `thread_limit()` clauses can be used to specify the number of teams and threads for each team

- `#pragma omp distribute <clause>`
  - Distribute specifies loops which are executed by the thread teams

- `#pragma omp distribute parallel for <clause>`
  - Same as distribute, but the loop iterations of each team are executed in parallel by the threads of each team
Acceleration support – GPU memory regions

- OpenMP supports different memory regions, also suited for those of GPUs
- The memory location is specified by using the shared and private clause
  - \texttt{shared (X)} = CUDA global memory
    - Accessible by all threads
  - \texttt{teams private (X)} = CUDA shared memory
    - Accessible by all threads in a team
  - \texttt{private (X)} = CUDA local memory
    - Each team member has own X
Acceleration support – code example
STREAM-ADD Benchmark on the GPU

#define N 262144

#pragma omp declare target
int add (int a, int b) {
    return a + b;
}
#pragma omp end declare target

int main ()
{
    int A[N];
    int *B;
    int x, y;
    B = malloc(N * sizeof(int));

    //fill array A and B with data
    //the compiler scans the target region for data accesses
    //A is an array of fixed size so the compiler can upload array A automatically
    #pragma omp target map(tofrom: B[0:N])
    {
        #pragma omp teams num teams(512) thread limit(256) private(x) // shared memory
        #pragma omp distribute parallel for private(y) // local memory
        for (int i = 0 ; i < N ; i++)
        {
            B[i] = add(A[i],B[i]);
        }
    //at the end of the target region array B gets downloaded from the device
    }
}
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GADGET</td>
<td>Brief introduction of the astrophysics code</td>
</tr>
<tr>
<td>OpenMP 4.0</td>
<td>What is new with version 4.0 of the OpenMP specification</td>
</tr>
<tr>
<td>LLVM C/C++ compiler with OpenMP 4.0 support</td>
<td>Presentation of the clang OpenMP compiler</td>
</tr>
<tr>
<td>Insights &amp; Experiences</td>
<td>Gathered about GADGET and offloading with OpenMP</td>
</tr>
<tr>
<td>Performance results</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
</tbody>
</table>
LLVM C/C++ compiler with OpenMP 4.0 support

Presentation of the clang OpenMP compiler
Clang is used as the front-end
- Parse source code and generate LLVM IR code
- Modified to generate code for OpenMP device constructs
- Produces two copies of code for target regions
- Insert calls to standardized OpenMP runtime interface functions
- Compiler driver modified to process code copies through different back-ends

NVPTX back-end
- Produces PTX code, then processed through ptxas to generate CUDA binary

Power back-end
- Preliminary performance

Ongoing development

https://github.com/clang-omp/clang

IBM contributed
# Table of contents

<table>
<thead>
<tr>
<th>GADGET</th>
<th>Brief introduction of the astrophysics code</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP 4.0</td>
<td>What is new with version 4.0 of the OpenMP specification</td>
</tr>
<tr>
<td>LLVM C/C++ compiler with</td>
<td>Presentation of the clang OpenMP compiler</td>
</tr>
<tr>
<td>OpenMP 4.0 support</td>
<td></td>
</tr>
<tr>
<td><strong>Insights &amp; Experiences</strong></td>
<td></td>
</tr>
<tr>
<td>Performance results</td>
<td>Gathered about GADGET and offloading with OpenMP</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
</tbody>
</table>
Insights & Experiences

...gathered about GADGET and offloading with OpenMP
Insights & Experiences – Algorithm

- Problem
  - The algorithm used by GADGET (tree-walk) is not well suited for execution on a GPU
  - Bodies take different paths in the octree leading to high warp divergence
  - In the original version the calculation is performed during the tree-walk
    - Increases the duration of divergent executions

- Action
  - The algorithm has been replaced by a so called vectorized Barnes-Hut algorithm
Insights & Experiences – Algorithm
vectorized Barnes-Hut

- Nearby bodies are grouped
  - tree-walk performed for groups
  - All bodies in group share one interaction-list

- Result
  - Less warp-divergence during the calculation
  - Fewer executions of the tree-walk
  - More calculations are performed

- In addition split tree-walk and calculation
  - First, tree-walk is performed (on the host)
  - Instead of immediately calculating interactions between bodies, store references to partners in an interaction-list
  - Second, interaction-forces are calculated from those references (on the GPU)

Grouping with N=3 leads to 5 groups
Insights & Experiences – Data Layout

- **Problem**
  - The data layout of GADGET is also not well suited for the GPU
  - GADGET stores the tree information and body data in arrays-of-structs
  - The data for internal and leaf nodes are stored in different arrays
    - Makes distinction between internal and leaf nodes in the calculation step necessary
    - Threads of a warp load data from different memory locations
    - Leading to a little bit of warp divergence

- **Actions**
  - Replace some of the array-of-structs by flat arrays
  - In interaction-list, replace reference to interaction-partner by interaction-partner’s data

- **Results**
  - Better coalescing, can still be improved
  - Distinction between internal and leaf nodes is only needed during the tree-walk
  - Very large interaction-lists, increased data-transfer times
  - Changes had nearly no impact on the performance of the calculation
Insights & Experiences – OpenMP 4.0

- With OpenMP 4.0 learning CUDA is not necessary to use a NVIDA GPU accelerator
- But instead one needs to learn new OpenMP pragmas
- CUDA provides the developer with finer control, for example pinned memory
- Simple variables and arrays of fixed size get uploaded to the device automatically
  - This advantage disappears with the new generation of GPUs
- As for CUDA, applications need to be reorganized to gain performance from using a GPU
  - Hardware properties of the GPU still need to be considered
    - Small cache, warp behavior, register count, shared memory, etc…
  - Likely the algorithm needs to be adjusted to avoid warp-divergence
  - Likely the data layout has to be reorganized to benefit from coalescing
Insights & Experiences – OpenMP compiler

- A lot of features are already supported although the compiler is in ongoing development
- The nvprof/nvvp profiler can be used to profile the device part of an OpenMP 4.0 application
- Some features have been implemented during the project
  - In the beginning all available shared memory got allocated for a block
    - Leading to only one block being executed per SM
    - Now the user can set the shared memory for each block with a compiler flag
- Some features are still missing
  - More debug information: nvvp unable to back-reference to source code
  - Currently the maximum amount of teams is limited to 512
  - Currently the compiler causes a relative large register usage overhead
- Some error messages are not very helpful
Table of contents

GADGET

Brief introduction of the astrophysics code

OpenMP 4.0

What is new with version 4.0 of the OpenMP specification

LLVM C/C++ compiler with OpenMP 4.0 support

Presentation of the clang OpenMP compiler

Insights & Experiences

Gathered about GADGET and offloading with OpenMP

Performance results

Summary
Performance Results
preliminary
Performance results – tree-walk
OpenMP parallelized

- Comments on GPU parts
  - Large local memory spilling (known compiler issue), impact vs. CUDA ~1/3
- VBH: still only 80% warp execution efficiency
- Adjusted data structures
  - Large interaction-list leads to increased data transport from the host to the GPU
  - Calculation time stays the same as before the change
  - Potential to coalesce even more
Performance results

Host-scaling

- POWER 8 memory-bandwidth not well-used
  - Latency bound, even though space-filling-curve to optimize tree-walk locality
Memory Bandwidth / Arithmetic Capacity

- tiny caches on GPU - space-filling-curve not required after all?
- Sufficient to walk the tree in groups?
Summary

- Original algorithm of GADGET has not been well suited for the execution on GPUs
- The same applies for the data layout of GADGET (array-of-structs)
- Some improvements in comparison to the native version have been achieved by changing the algorithm and data layout
- Still a factor of 1.6 slower than the CPU optimized version
  - Warp divergence can not be fully avoided as branching is part of the calculation
  - CPU version unable to leverage full memory bandwidth of the Power 8 processor
- Developers do not need CUDA to accelerate their applications with GPUs
- A lot of features already supported with the current version of the compiler
- For some applications only adding pragmas into the CPU optimized code will not yield good performance on the GPU
References & Sources

- http://physics.princeton.edu/~fpretori/Nbody/intro.htm
- http://wwwmpa.mpa-garching.mpg.de/gadget
- http://arborjs.org/docs/barnes-hut
- https://www.scai.fraunhofer.de/fileadmin/ArbeitsgruppeTrottenberg/SS08/seminar/Vortrag.pdf
- http://openmp.org/mp-documents/OpenMP-4.0-C.pdf
- http://openmp.org/sc14/Booth-Sam-IBM.pdf