Parallel Architectures and Programming in Virtual Arabia

Alin Murarasu
murarasu@in.tum.de
Outline

- The Sparse Grid Benchmark
- Optimizations
- CPU vs. GPU
- From Benchmark to Library
- Hybrid Programming
- Current Status & Next Steps
The Sparse Grid Benchmark

- Composed of 2 kernels: compression (hierarchization) and decompression (evaluation) – 500 code lines
- *Focus is on performance, not on approximation quality*
- Simplifications: no real data, 0-boundary functions
- It enabled us to experiment with:
  - Data structures: tree, hash-table, trie, *bijection-based*
  - Algorithms: recursive and *iterative*
  - Programming models: OpenMP, MPI, CUDA, StarPU
  - Computer architectures: multi-core CPUs and GPUs
    - *Wide range of optimizations*
- We evaluated various hardware: 4-, 8-, and 12-core Nehalem, 32-core Opteron, Tesla C1060
Optimizations

- Our algorithmic optimizations are based on a bijection that maps the points of a sparse grid to a set of consecutive natural numbers
  - Minimal memory consumption (space optimization)
  - Reduced number of cache misses (time optimization)
  - “Compact Data Structures and Parallel Algorithms for the Sparse Grid Technique”, PPoPP'2011
CPU vs. GPU

- CPU = general-purpose, low latency, high frequency processor, 4 – 6 cores, 4-way SIMD, out-of-order execution, branch prediction, data prefetching, etc.
- GPU = special-purpose, high throughput, low frequency processor, 16 – 30 cores, 2x16- and 8-way SIMD, in-order execution, extreme multithreading
- CPU optimizations: cache, SSE
- GPU optimizations: use of shared memory and constant cache, fewer bank conflicts, more active threads, coalesced memory accesses
- Our benchmark revealed GPU as the winning platform
Dual-layer design:

1) Data structure: structures and functions for accessing data
2) Sparse grid operations: hierarchization and evaluation

Dimensional adaptivity is a form of adaptivity (coarse) that maintains the regular structure of the sparse grid

Supporting non-0 boundary functions is essential; for this, we employ a projection based approach
Hybrid Programming

- Example of hybrid machine: CPU + GPU
- This is a distributed memory system (PCIe is a bottleneck)
- An 8-core Nehalem with Hyperthreading provides:
  - 10x for compression
  - 12x for decompression
- An Nvidia Tesla C1060 provides:
  - 17x faster compression
  - 70x faster decompression
- Can we obtain ~27x and ~82x from this hybrid system?
  - No for compression due to irregular accesses
  - Yes for decompression (up to 98% efficiency)
- The library should harness the processing power available in heterogeneous systems
- **Load balancing is a focus point for us**
- Other points of interest: programming models and performance analysis
Current Status & Next Steps

- We have developed a **benchmark** which we use for assessing the performance of various architectures.
- We are currently in the process of converting the benchmark into a **fully functional, optimized library**.
- For **load balancing in heterogeneous system**, we are considering extensions to the StarPU framework.
- We plan to port our benchmark on Nvidia Fermi and AMD GPUs.
Thank you for listening!