Center for Computational Sciences, University of Tsukuba
www.ccs.tsukuba.ac.jp

High Performance Computing Research

Quadruple Precision BLAS on GPUs

Background

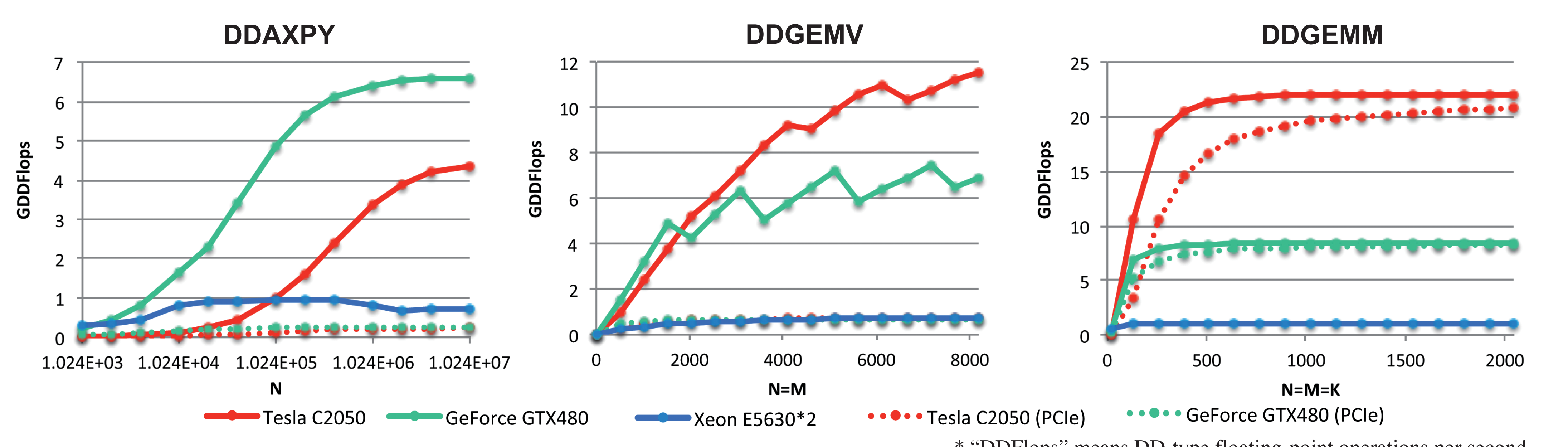
- There is high demand for high precision floating-point operations (e.g. reducing error accumulation in large scale computing)
- Software emulation of high precision operation are extremely computationally complex: it severely limits the performance on most traditional computer systems
- GPUs have higher peak performance than CPUs, and GPUs are suitable for vector operations
- We expect that the high precision operations of linear algebra on GPUs attain higher performance than that on CPUs

Overview

- We implemented quadruple precision Basic Linear Algebra Subprograms (BLAS) using CUDA for NVIDIA's GPUs
- We used double-double (DD) type quadruple precision operations [Bailey et al.]
- DD-type operations store a quadruple precision value into two double precision floating point values, and emulate quadruple precision operations using double precision operations

Performance

- On GPUs, the quadruple precision BLAS subroutines are much faster than that on CPUs
- DDAXPY and DDGEMV are memory-bound on Tesla C2050. Thus, the execution times of DDAXPY and DDGEMV are approx. 2 times more than that of DAXPY and DGEMV on CUBLAS 4.0, respectively



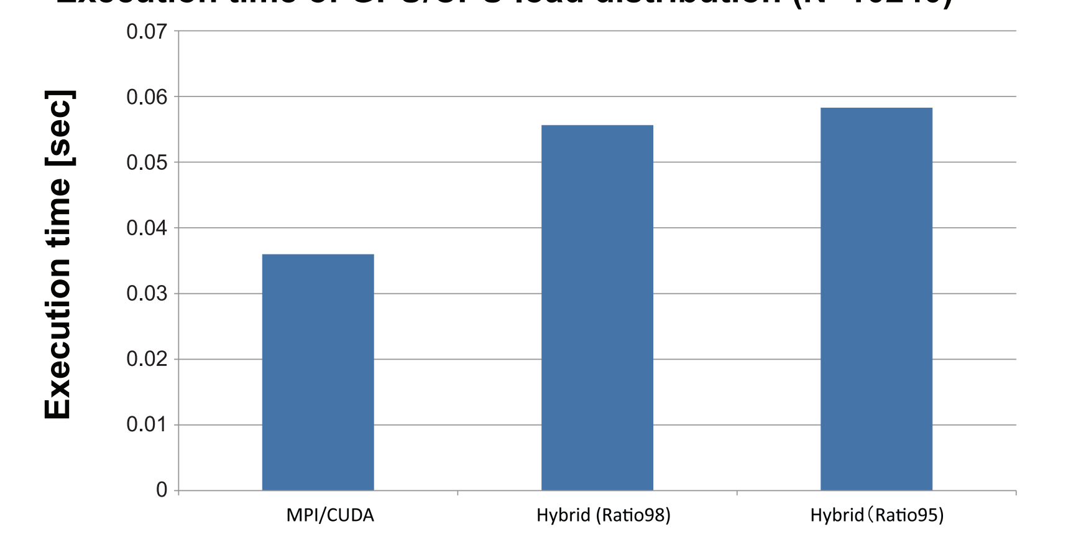
* "DDFlops" means DD-type floating-point operations per second

CPU/GPU Hybrid Co-Working With Easy Programming

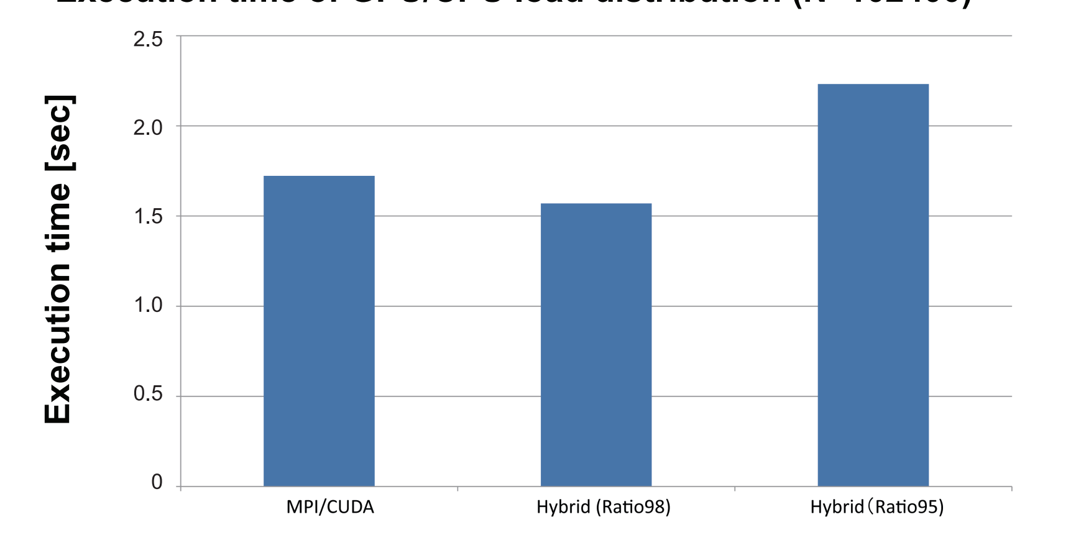
- **Motivation:** GPU clusters are generally used in the style of "function acceleration" to dispatch heavy computation parts to GPU and CPU only takes light-weight computation and MPI communication. To exploit the full performance of GPU and CPU in each node, it is required to distribute the computation load to both of them in well-balanced manner.
- **Experiments:** For a simple N-body computation in astrophysics, the computation load is distributed to multi-core CPU and GPU in every time step. "Ratio" is the fraction of computation load for GPU. It is shown that the best balance depends on the problem size where I/O load is in O(N) while computation load is in O(N2).
- **Solution:** It is required to make a good balance between GPU and CPU loads, and it is difficult to control statically. To avoid programming difficulty, we combine XcalableMP-dev language and StarPU process management system, which are developed in University of Tsukuba and INRIA Bordeaux, respectively. The variables and loops indicated by special directives are automatically mapped on data pool and dynamic processes in StarPU by XcalableMP-dev compiler.

* This research is supported by Japan-France joint project named "Framework and Programming for Post Petascale Computing" by JST-ANR.

Execution time of GPU/CPU load distribution (N=10240)



Execution time of GPU/CPU load distribution (N=102400)

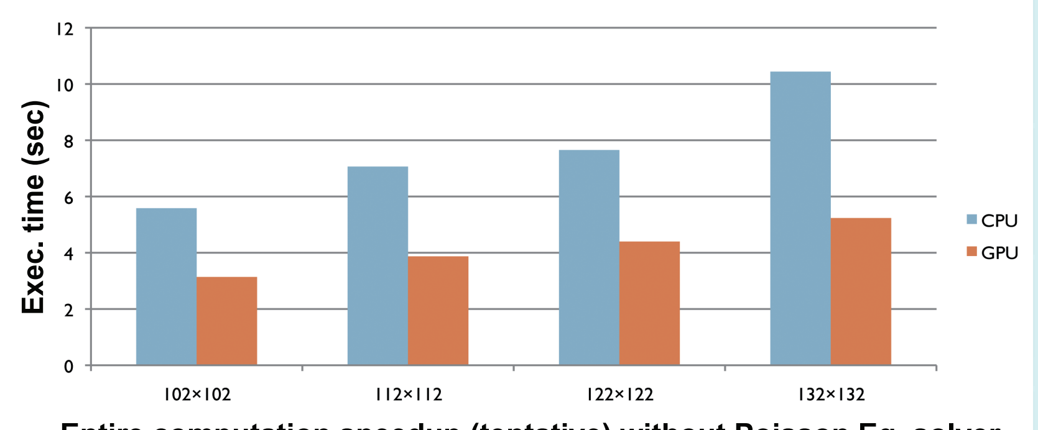


Large Eddy Simulation on GPU

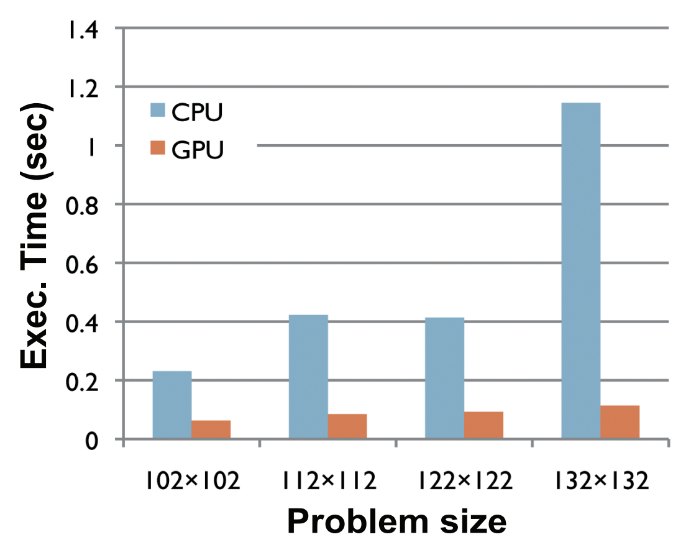
LES (Large Eddy Simulation) on city-level climate simulation with various physics and chemistry process is a computation intensive problem suitable for GPU computing. We are developing a very high resolution of LES code for large scale GPU clusters such as HA-PACS, reducing CPU-GPU communication overhead to merge small functions into larger GPU kernel function. Currently, all the functions except Poisson solver which consumes approximately 30% of entire computation, are merged into just one function to exploit high performance. Implementing additional features for building structure analysis and other physical reactions, we will complete the implementation of "city model" LES with fine resolution driven by large-scale parallel GPU computations.

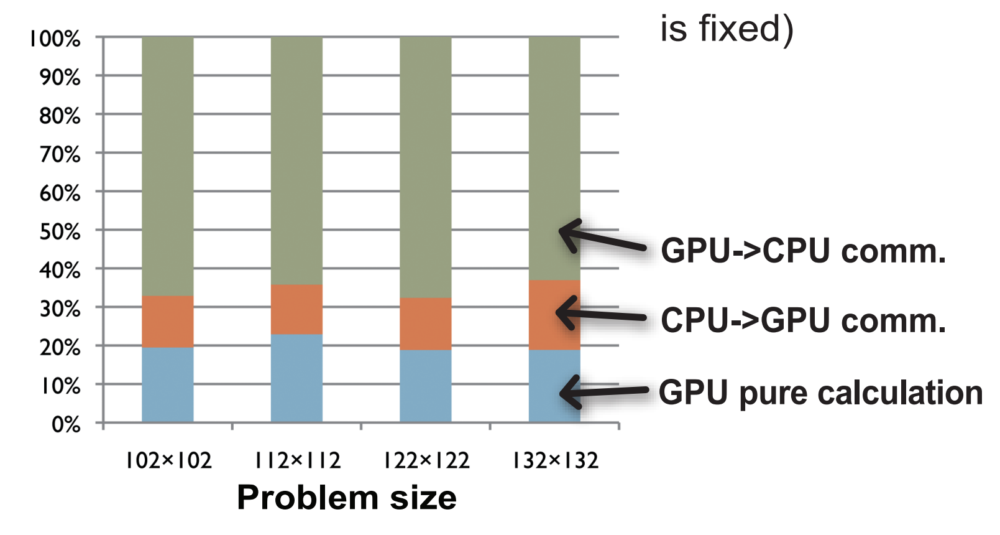
Testing environment

Item	Spec.
CPU	Intel Xeon E5630 2.53GHz 4cores × 2
RAM	DDR3 SDRAM 1066MHz 4GB × 6 GDDR 5 SDRAM 1.55GHz 3GB (ECC on)
GPU	NVIDIA Tesla M2050 1.15GHz
OS	CentOS Linux release 6.0 (Final)
Compiler	GNU Fortran (GCC) 4.4.4 nvcc 4.0 (-arch sm_20) for GPU code



Entire computation speedup (tentative) without Poisson Eq. solver





Execution time comparison on CPU and GPU including I/O (left) and breakdown of GPU computation (right), for "cell surface gradient calculation" (size is shown as grid size of X x Y where Z-dimension size is fixed)