Tapas: An Implicitly Parallel Programming Framework For Hierarchical N-body Algorithms

Keisuke Fukuda*†§, Motohiko Matsuda*, Naoya Maruyama*, Rio Yokota†, Kenjiro Taura‡ and Satoshi Matsuoka†
*RIKEN Advanced Institute for Computational Science, Hyogo, Japan
†Tokyo Institute of Technology, Tokyo, Japan
‡University of Tokyo, Tokyo, Japan
§Email:fukuda@matsulab.is.titech.ac.jp

Exascale computing places significant research challenges in developing algorithms that can effectively exploit the capabilities of future computing systems [6]. One possible answer to the challenge is hierarchical algorithms thanks to their lower computational complexity, arithmetic density, and communication locality. Such algorithms include Barnes-Hut, the Fast Multipole Method (FMM), and $\mathcal{H} - matrix$.

Developing efficient implementations of such hierarchical algorithms for today's and future supercomputing systems, however, is a non-trivial and labor-intensive task. Especially for application scientists, scalable implementations has been challenging due to the irregular and complex data structure and inherent dynamic data dependencies.

There have been substantial efforts to ease the development of efficient scientific applications with high-level programming abstractions [3–5, 7, 8, 10, 12]. However, there has been little work that addresses the problem in the context of irregular algorithms such as FMM. Most of the existing FMM implementations are developed from scratch with very limited use of software engineering discipline [1, 2, 9, 13], leading to poor productivity.Although there have been efforts to couple more software engineering with FMM, the lack of performance comparisons of these codes against the hand-tuned state-of-the-art [1, 2, 9, 13] makes it difficult to assess the overhead of introducing such middleware into FMM codes.

We address the problem of application development productivity for a class of hierarchical N-body algorithms represented by, but not limited to, FMM. Specifically, we aim to realize a programming interface that allows separation of concerns between algorithmic and architectural issues such that a single implementation of a hierarchical N-body algorithm, once written, can efficiently run on different parallel architectures. To that end, we propose *Tapas*, a high-level implicitly parallel programming framework for hierarchical N-body algorithms. This poster presents a prototype implementation of Tapas using C++ template metaprogramming and its evaluations. Despite implemented solely using standard C++ language features, the Tapas prototype allows for automatic parallelization over distributed memory machines and shared-memory, multi-core CPUs. Unlike regular computations such as stencils, data dependencies in our target problem domain are not statically determined, requiring dependence analysis at runtime. The key novelty in our framework is that the dependency issue is automatically resolved by a transparent inspector-executor method [11], allowing automatic parallelization even for distributed memory environments. Furthermore, compute-intensive kernels such as direct computations are automatically offloaded to CUDA-based GPU accelerators. The fact that it does not use any custom compilers or source-to-source translators except for standard C++ features has a practical advantage in the sense that it can be safely assumed to run on almost all major current and future platforms.

Tapas provides several abstractions for hierarchical n-body applications, which enable automatic parallelization of implicit-style parallel code written by the programmer. The core data types in Tapas are `Body` and `Cell`, which represents body and tree nodes in the simulation. The programmer can define custom attributes to `Body` and `Cell`, which hold application-specific data. The programmer also defines C++ template functors, that define the application's tree traversal operations. The operations are defined using C++ template so that the Tapas framework can generate an inspector and executor for inter-process communication. For traversal operation functors, Tapas provides two major functions, `Map` and `Reduce` to compute attribute values of the tree structure. `Map` applies a functor to children of a cell and `Reduce` accumulates values from children to the parent. Fig. 1 and Fig. 2 show simplified code examples.

To evaluate the Tapas prototype, we implement the same FMM algorithm as that of the ExaFMM library [13], one of the fastest known implementations of FMM to date, and compare their performances using a GPU-based heterogeneous supercomputer. We show that Tapas can achieve comparable performance as the hand-tuned FMM code and scale thousands of CPU cores while efficiently using GPUs. More specifically, our FMM implementation delivers 1.15x faster over ExaFMM in serial execution. It also shows good strong scalability up to 1536 cores. The multi-GPU version achieves a 2.5x speedup over the CPU version when executed on 100 nodes of TSUBAME2.5 with 300 GPU.

ACKNOWLEDGMENTS

This project was partially supported by JST, CREST through its research program: "Highly Productive, High Performance

```
1  // Post-order tree traversal functor
2  struct TraversePostOrder {
3    template<typename Cell>
4    void operator()(Cell &p, Cell &c) {
5      // p is the parent, c is a child
6      // To update parent, use Reduce() API
7      // Reduce values from all the children
8      // using reducing function 'sum'.
9      // The field name 'foo' is defined by the programmer.
10     Reduce(p. p.attr().foo, val, sum);
11
12     Map(TraversePostOrder(), c.SubCells());
13     ...; // do something using C;
14   }
15 };
16 void main() {
17   Cell root; // Tree root cell
18   // Start a post-order traversal
19   Map(TraversePostOrder(), root);
20 }
```

Fig. 1. Simplified example code snippets of pre-order and post-order traversals.

```
1  // Dual tree traversal functor
2  struct DualTreeTraversal {}
3    template<typename Cell>
4    inline void operator()(Cell C1, Cell C2) {
5      ...; // do something using C1 and C2
6      Map(DualTreeTraversal(),
7          Product(C1.SubCells(), C2.SubCells());
8    }
9  };
10 void main() {
11   Cell root; // Tree root cell
12   // Start a dual tree traversal
13   Map(DualTreeTraversal(), root, root);
14 }
```

Fig. 2. Simplified example code snippets of dual-tree traversal.

Application Frameworks for Post Petascale Computing" and "Advanced Core Technologies for Big Data Integration", as well as JSPS KAKENHI Grant Number 16H02827.

REFERENCES

[1] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, and T. Takahashi, "Task-based fmm for multicore architectures," *SIAM Journal on Scientific Computing*, vol. 36, no. 1, pp. C66–C93, 2014.

[2] J. Bédorf, E. Gaburov, M. S. Fujii, K. Nitadori, T. Ishiyama, and S. Portegies Zwart, "24.77 Pflops on a gravitational tree-code to simulate the milky way galaxy with 18600 GPUs," in *Proceedings of the 2014 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 1–12.

[3] B. Chamberlain, D. Callahan, and H. Zima, "Parallel programmability and the chapel language," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 3, pp. 291–312, Aug. 2007.

[4] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar, "X10: An object-oriented approach to non-uniform cluster computing," in *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA '05. ACM, 2005, pp. 519–538.

[5] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[6] DOE ASCAC Subcommittee, "Top ten exascale research challenges," February 2014.

[7] K. Gregory and A. Miller, *Accelerated Massive Parallelism with Microsoft Visual C++*. Microsoft Press, September 2012.

[8] T. Gysi, C. Osuna, O. Fuhrer, M. Bianco, and T. C. Schulthess, "Stella: A domain-specific tool for structured grid methods in weather and climate models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. ACM, 2015, pp. 41:1–41:12.

[9] D. Malhotra and G. Biros, "PVFMM: A parallel kernel independent FMM for particle and volume potentials," *Communications in Computational Physics*, vol. 18, no. 3, pp. 808–830, 2015.

[10] N. Maruyama, T. Nomura, K. Sato, and S. Matsuoka, "Physis: An implicitly parallel programming model for stencil computations on large-scale gpu-accelerated supercomputers," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. ACM, 2011, pp. 11:1–11:12.

[11] J. H. Saltz, R. Mirchandaney, and K. Crowley, "Run-time parallelization and scheduling of loops," *IEEE Transactions on Computers*, vol. 40, no. 5, pp. 603–612, May 1991.

[12] A. K. Sujeeth, K. J. Brown, H. Lee, T. Rompf, H. Chafi, M. Odersky, and K. Olukotun, "Delite: A compiler architecture for performance-oriented embedded domain-specific languages," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4s, pp. 134:1–134:25, Apr. 2014.

[13] R. Yokota and L. A. Barba, "A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems," *International Journal of High Performance Computing Applications*, vol. 26, no. 4, pp. 337–346, 2012.