Orthotope Machine

Takayuki Muranushi

April 10, 2011

In geometry, an orthotope (also called a hyperrectangle or a box) is the generalization of a rectangle for higher dimensions, formally defined as the Cartesian product of intervals.

1 Introduction

This document describes the *Orthotope Machine*, a virtual machine that operates on multidimensional arrays. The Orthotope Machine is one of the main components for Paraiso project. The goal of Paraiso project is to create a high-level programming language for generating massively parallel, explicit solver algorithms of partial differential equations.

From computational viewpoint, explicit solvers of partial differential equations belongs to the algorithm category called stencil codes. Stencil codes are algorithms that updates the array, each element accessing the nearby elements in the same pattern (c.f. Fig.1). Stencil codes are commonly used algorithms in fields such as solving partial differential equations and image processing. Code generations and automated tuning for stencil codes has been studied e.g. [Dat09, DMV+08].

There are many methods other than stencil codes for solving partial differential equations. They have different merits. A notable project in progress is Liszt [CDM+], an embedded DSL in programmin language Scala, designed for generating hydrodynamics solver on unstructured mesh.

Many parallel and distributed programming languages has been implemented using Haskell [TLP02]. Data Parallel Haskell [PJ08] and Nepal[CKLP01] are implementations of NESL, a language for operating nested arrays. Accelerate [CKL+11] and Nikola [MM10] are languages to manipulate arrays on GPUs written in Haskell.

```
double a[NY][NX], b[NY][NX];
for (int t=0; t<max_t; ++t) {
  for (int y=1; y<NY-1; ++y)
    for (int x=1; x<NX-1; ++x)
      b[y][x] = a[y][x-1] + a[y][x+1]
              + a[y-1][x] + a[y+1][x];

  for (int y=1; y<NY-1; ++y)
    for (int x=1; x<NX-1; ++x)
      a[y][x] += 0.25 * b[y][x];
}
```

Figure 1: An example of stencil code.

We need new languages for parallel hardwares — this is a long-standing idea. Many project sought for them, and some failed. Failures from which we can learn. High Performance Fortran was a very promising approach to introduce a high-level parallelism in Fortran but, as James Stone told me in Taiwan, and as is summarised by the project leader [KKZ07], it failed. DEQSOL [NCY15, KSSU86] was another project which had design similar to that of Paraiso. The language was initially designed for Hitachi vector machines. The extension of DEQSOL for parallel vector machines has been planned [NY15] but seemingly did not realize.

The unique point of Paraiso compared to those projects is its focus on computational domains that utilize localized access to multidimensional arrays.

Multidimensional arrays are different from nested arrays. For example in the psueudocode Fig.1, in order to calculate `b[y][x]` you need to read from `a[y-1][x]` and `a[y+1][x]`, which are usually located much farther in the memory compared to `a[y][x-32]` or `a[y][x+64]`. The code generator must be aware of such locality in multidimensional space. For most of the cases, the basic equations to be solved is symmetric under exchange of the axes (X,Y,Z ...). Still, there are nonnegligible differences between the axes from computational point of view, especially if the multidimensional arrays are stored in row-major or column-major order. To utilize the cache and/or vector instructions, we need to know, or decide upon, the order the array is stored in the memory.

In parallel machines, the array must be decomposed and distributed among computer nodes. It is important to take care of the continuity in multidimensional space when making the distribution, so that the communications cost is lowered. If the data to be communicated is stored sparsely in the memory, it is a good strategy to gather them manually into a single buffer and to make a single large communication instead of making lots of small communications. Due to this gather/scattering cost, not making any decomposition in one or more directions gives higher performance in some cases.

The Orthotope Machine is designed to capture and utilize these characteristics of the multidimensional array computations.

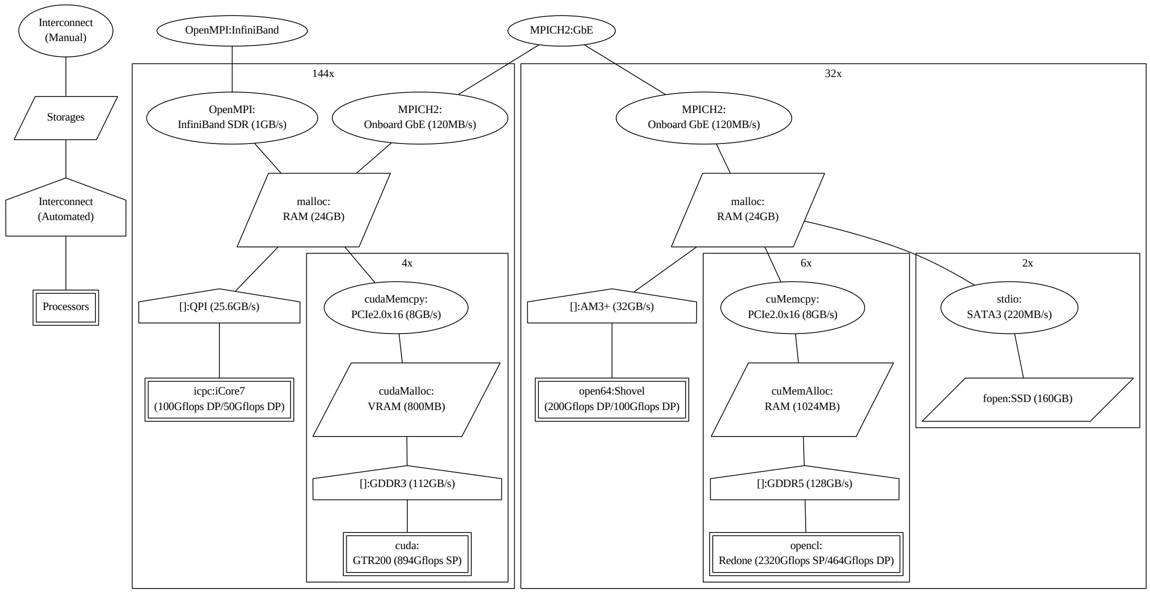
2 Explicit Solvers of Partial Differential Equations

Fig. 2 shows a pseudocode for a hydrodynamic equations solver. Although it is a bit simplified from the real solvers, it shows what components needed to build one.

A common task for the solver is to simulate the evolution of the fluid for a certain interval of time, say, $0 < t < t_{max}$ (1). In the each step of the simulation, the time `t` increases by a certain amount `dt`. In many problems, the timestep `dt` is not a constant but depends on the state of the fluid. In such case, you need to calculate the timesteps adequate for the fluid state of every mesh (2), and then perform reduction over the entire computational domain to calculate the smallest `dt` (3).

```
double fluid[NZ][NY][NX];
double flow_x[NZ][NY][NX];
double flow_y[NZ][NY][NX];
double flow_z[NZ][NY][NX];
double dt_local[NZ][NY][NX];

// (1) simulation goes from time t=0 to t=t_max
for (double t=0; t<t_max; t+=dt) {

  // (2) calculate the timescale for each mesh
  for (int z=1; y<NZ-1; ++z)
    for (int y=1; y<NY-1; ++y)
      for (int x=1; x<NX-1; ++x)
        dt_local[z][y][x]=timescale(fluid[z][y][x]);

  // (3) calculate the minimum timescale
  double dt=max_t;
  for (int z=1; y<NZ-1; ++z)
    for (int y=1; y<NY-1; ++y)
      for (int x=1; x<NX-1; ++x)
        dt=min(dt, dt_local[z][y][x]);

  // (4) calculate the flow for each direction
  for (int z=1; y<NZ; ++z) {
    for (int y=1; y<NY; ++y) {
      for (int x=1; x<NX; ++x) {
        flow_x[z][y][x]=calc_fx(fluid[z][y][x-1], fluid[z][y][x]);
        flow_y[z][y][x]=calc_fy(fluid[z][y-1][x], fluid[z][y][x]);
        flow_z[z][y][x]=calc_fz(fluid[z-1][y][x], fluid[z][y][x]);
      }
    }
  }

  // (5) move the fluid according to the flow
  for (int z=1; y<NZ-1; ++z)
    for (int y=1; y<NY-1; ++y)
      for (int x=1; x<NX-1; ++x)
          fluid[z][y][x] +=
          dt*(flow_x[z][y][x]-flow_x[z][y][x+1])/dx
         +dt*(flow_y[z][y][x]-flow_x[z][y+1][x])/dy
         +dt*(flow_z[z][y][x]-flow_x[z+1][y][x])/dz;

}
```

Figure 2: A typical hydrodynamic equations solver does something like this.



3 Overall Design of Paraiso, and Orthotope Machine's Role in it

4 Definitions of Orthotope, Orthotree, and Distributed Orthotope

5 API for Orthotope Machine

6 Hardware Model

7 Instruction Set

7.1 Instruction Set for Primodial Orthotope Machine

7.2 Instruction Set for Distributed Orthotope Machine

8 Possible Program Transformations

8.1 Common Techniques

8.2 Timestep Fusion

8.3 Cache

8.4 Data-Flow-Tree Manipulations by User Specification

8.5 Synchronization Insertion

8.6 Trapezium Splitting

jeeth, Pat Hanrahan, Martin Odersky, and Kunle Olukotun. Language virtualization for heterogeneous parallel computing. *SIGPLAN Not.*, 45:835–847.

[CKL+11] Manuel M.T. Chakravarty, Gabriele Keller, Sean Lee, Trevor L. McDonell, and Vinod Grover. Accelerating haskell array codes with multicore gpus. In *Proceedings of the sixth workshop on Declarative aspects of multicore programming*, DAMP '11, pages 3–14, New York, NY, USA, 2011. ACM.

[CKLP01] Manuel Chakravarty, Gabriele Keller, Roman Lechtchinsky, and Wolf Pfannenstiel. Nepal nested data parallelism in haskell. In Rizos Sakellariou, John Gurd, Len Freeman, and John Keane, editors, *Euro-Par 2001 Parallel Processing*, volume 2150 of *Lecture Notes in Computer Science*, pages 524–534. Springer Berlin / Heidelberg, 2001.

[Dat09] Kaushik Datta. *Auto-tuning Stencil Codes for Cache-Based Multicore Platforms*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2009.

[DMV+08] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 4:1–4:12, Piscataway, NJ, USA, 2008. IEEE Press.

[KKZ07] Ken Kennedy, Charles Koelbel, and Hans Zima. The rise and fall of high performance fortran: an historical object lesson. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, HOPL III, pages 7–1–7–22, New York, NY, USA, 2007. ACM.

[KSSU86] Chisato Kon'no, Miyuki Saji, Nobutoshi Sagawa, and Yukio Umetani. Advanced implicit solution function of deqsol and its evaluation. In *Proceedings of 1986 ACM Fall joint computer conference*, ACM '86, pages 1026–1033, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[MM10] Geoffrey Mainland and Greg Morrisett. Nikola: embedding compiled gpu functions in haskell. In *Proceedings of the third ACM Haskell symposium on Haskell*, Haskell '10, pages 67–78, New York, NY, USA, 2010. ACM.

[NCY15] SAGAWA NOBUTOSHI, KON'NO CHISATO, and UMETANI YUKIO. Numerical simulation language deqsol. *Transactions of Information Processing Society of Japan*, 30(1):36–45, 1989-01-15.

[NY15] Sagawa Nobutoshi and Umetani Yukio. Basic research on deqsol for parallel processors. *Zenkoku Taikai Kouen Ronbun Shu*, 38(3):1492–1493, 1989-03-15.

[PJ08] Simon Peyton Jones. Harnessing the multicores: Nested data parallelism in haskell. In G. Ramalingam, editor, *Programming Languages and Systems*, volume 5356 of *Lecture Notes in Computer Science*, pages 138–138. Springer Berlin / Heidelberg, 2008.

[TLP02] P. W. TRINDER, H.-W. LOIDL, and R. F. POINTON. Parallel and distributed haskells. *Journal of Functional Programming*, 12(4-5):469–510, 2002.