Supercomputing: DataFlow vs. ControlFlow (Paradigm Shift in Both, Benchmarking Methodology and Programming Model)

Milutinovic, V., Salom, J., Trifunovic, N., and Valero, M.

Introductory frame #1:

This paper was prepared in response to over 100 e-mail messages with questions from the CACM readership inspired by our previous CACM contribution [1].

Introduction

The strength of DataFlow computers, compared to ControlFlow ones, is in the fact that they accelerate the data flows and application loops for one or more orders of magnitude; how many orders of magnitude – that depends on the amount of data reusability within the loops. This feature is enabled by compiling down to levels much below the machine code, which brings important additional effects: much lower execution time, equipment size, and power dissipation.

The previous paper [1] argues that time has come to redefine Top 500 benchmarking. Concrete measurement data from real applications in GeoPhysics (Schlumberger) and FinancialEngineering (JPMorgan), shows that a DataFlow machine (for example, the Maxeler MAX series) rates better than a ControlFlow machine (for example, Cray Titan), if a different benchmark is used (e.g., a BigData benchmark), as well as a different ranking methodology (e.g., the benchmark execution time multiplied by the number of 1U boxes needed to achieve the given execution time - it is assumed, no matter what technology is inside, the 1U box always has the same size and always uses the same power).

In reaction to the previous paper [1], scientific community insists that more light is shed on two issues: (a) Programming paradigm and (b) Benchmarking methodology. Consequently the stress of this viewpoint is on these two issues.

Programming Model

What is the fastest, the least complex, and the least power consuming way to do computing?

Answer: Rather than writing one program to control the flow of data through the computer, one has to write a program to configure the hardware of the computer, so that input data, when it arrives, can flow through the computer hardware in only one way (the way how the computer hardware has been configured). This is best achieved if the serial part of the application (the transactions) continues to run on the ControlFlow host and the parallel part of the application (BigData crunching and loops) is migrated into a DataFlow accelerator.

The early works of Dennis [2] and Arvind [3] could prove the concept, but could not result in commercial successes for three reasons: (a) Reconfigurable hardware technology was not yet ready (contemporary

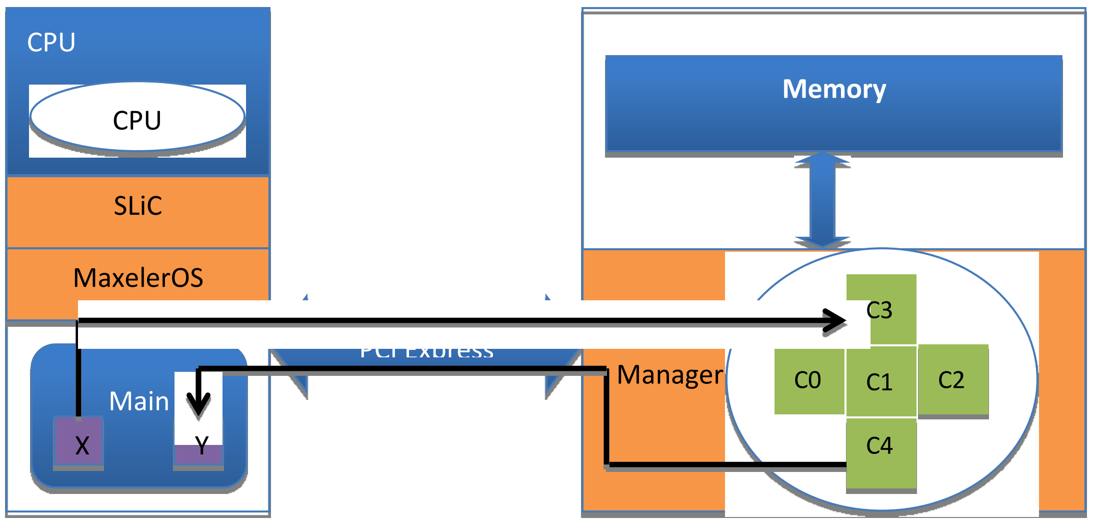
ASIC was fast enough but not reconfigurable, while reconfigurable FPGA was nonexistent); (b) System software technology was not yet ready (methodologies for fast creation of system software did exist, but effective tools for large scale efforts of this sort did not); and (c) Applications of those days were not of the BigData type, so the streaming capabilities of the DataFlow computing model could not generate performance superiority (recent measurements show, that, currently, Maxeler can move internally over 1TB of data per second [4]).

Each programming model is characterized with its quantity and quality. The quantity and quality aspects of the Maxeler DataFlow model are discussed in the next two paragraphs, based on Figure 1.

Quantitatively speaking, the complexity of DataFlow programming, in the case of Maxeler, is equal to 2n + 3, where n refers to the number of loops migrated from the ControlFlow host to the DataFlow accelerator. This means, the following programs have to be written:

- One kernel program per loop, to map the loop onto the DataFlow hardware;
- One kernel test program per loop, to test the above;
- One manager program (no matter how many kernels there are) to move data:
    - (a) Into the DataFlow accelerator,
    - (b) In between the kernels (if more than one kernel exists), and
    - (c) Out of the DataFlow accelerator;
- One simulation builder program,
  to test code without the time consuming migration into the binary level;
- One hardware builder program, to exploit the code on the binary level.

In addition, in the host program (initially written in Fortran, Hadoop, MapReduce, MathLab, Matematika, C++, or C), instead of each migrated loop, one has to include a streaming construct (send data + receive results).



CPU Code (.c)

```
#include "MaxSLiCInterface.h"
#include "Calc.max"

int *x, *y;

Calc(x, DATA_SIZE);
```

Manager (.java)

```
Manager m =
   new Manager("Calc");
Kernel k = new
MyKernel();
m.setKernel(k);
m.setIO(
    link("x", PCIE),
    link("y", PCIE));
m.addMode(modeDefault());
m.build();
```

MyKernel (.java)

```
DFEvar x = io.input("x", hwInt(32));

DFEvar result = c0*FIFO(x,1) +
                c1*x +
                c2*FIFI(x,-1) +
                c3*FIFI(x, L) +
                c4*FIFI(x,-L);
```

Figure 1: An example Host code, Manager code, and Kernel code (a single kernel case): 2D Convolution

Legend:
SLIC = Compiler support for selected domain specific languages and customer applications
DFEvar = keyword used for defining the variables that internally flow through the configured hardware
(in contrast to standard Java variables used to instruct the compiler)

Qualitatively speaking, the above quantity (2n + 3) is not any more difficult to realize because of the existence of a DSL (domain specific language) like MaxJ (an extension of standard Java with over 100 new functionalities). Figure 2 shows that a relatively complex BigData processing problem can be realized in only a few lines of MaxJ code. Note that the programming model implies the need for existence of two types of variables: (a) Standard Java variables, to control compile time activities, and (b) DFE (Data Flow Engine) variables, which actually flow through configured hardware (denoted with the DFE prefix in the examples of figures 1 and 2).

HostCode (.c)

```
for(t = 1; t < tmax; t++) {
  // Set-up timestep
  if(t < tsrc) {
    source = generate.soruce.wavelet(t);
    maxlib.stream.region_from_host(
                  maxlib, "source", source
                  srcx, srcy, srcz,
                  srcx+1, srcy+1, srcz+1);
  }
  maxlib.stream_from_dram(maxlib, "curr",
curr_ptr);
  maxlib.stream_from_dram(maxlib, "prev",
prev_ptr);
  maxlib.stream_earthmodel_from_dram(maxlib,
dvv_array);

  maxlib.stream_to_dram(maxlib, "next", next_ptr);

  maxlib_run(maxlib); // Execute timestep

  swap_buffers(prev_ptr, curr_ptr, next_ptr);
 }
```

FDKernel (.java)

```
public class IsotropicModelingKernel extends FDKernel {
   public IsotropicModelingKernel(FDKernelParameters p) {
        super(p);
        Stencil stencil =
           fixedStencil(-6, 6, coeffs, 1/8.0);
        DFEVar curr = io.wavefieldInput("curr", 1.0, 6);
        DFEVar prev = io.wavefieldInput("curr", 1.0, 0);
        DFEVar dvv = io.earthModelInput("dvv", 9.0, 0);
        DFEVar source = io.hostInput("source", 1.0n 0);

        DFEVar I =
           convolve(curr, ConvolveAxes.XYZ.stencil);

        DFEVar next = curr*2 - prev + dvv + I + source;

        io.wavefieldOutput("next", next);
   }
}
```

Figure 2: An example of the Host and Kernel Code
(The programming is largely facilitated through the use of appropriate Java extensions)

Legend:
DFEvar = keyword used for defining the variables that internally flow through the configured hardware
(in contrast to standard Java variables used to instruct the compiler)

Benchmarking Methodology

Bare speed is definitely neither the only issue of importance nor the most important one [5]. Consequently, the Top 500 ranking should not concentrate on only one issue of importance, no matter if it is speed, power dissipation, or size (the size includes hardware complexity in the widest sense); it should concentrate on all three issues together, at the same time.

In this paper we argue that the best methodology for Top 500 benchmarking should be based on the holistic performance measure H ($T_{BigData}$, N_{1U}) (HPM) defined as the number of 1U boxes (N_{1U}) (or equivalent) needed to achieve the desired execution time using a given BigData benchmark. Issues like power dissipation (monthly electricity bill), and the physical size of the equipment (the assumption here is that equipment size is proportional to the hardware complexity, and to the hardware production cost) are implicitly covered by H ($T_{BigData}$, N_{1U}). Selection of the performance measure H is coherent with the TPA concept introduced in [6] and described in Figure 3.

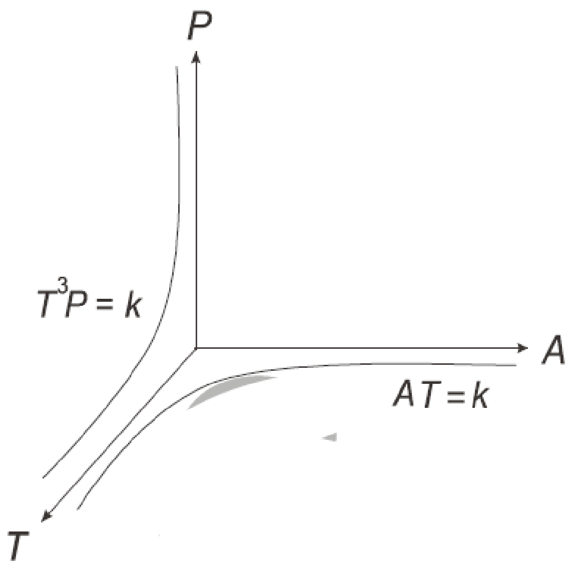


Figure 3: The TPA (Time, Power, and Area) Concept of the Optimal Computer Design
Design optimizations have to optimize the three essential issues jointly:
T = Time, P = Power, and A = Area (complexity of a VLSI chip).

Note that the hardware design cost is not encompassed by the parameter A (parameter A encompasses only the hardware production cost), which causes that the above defined H formula represents an upper bound for ControlFlow machines and a lower bound for DataFlow machines. This is due to the fact that ControlFlow machines are based on the Von Neumann logic, which is complex to design (execution control unit, cash control mechanism, prediction mechanisms, etc.), while the DataFlow machines are based on the FPGA logic, which is simple to design (mostly because the level of design repetitiveness is extremely high, etc…).

As indicated in the previous paper [1], the performance measure H puts PetaFlops out of date, and brings PetaData into the focus. Consequently, if the Top 500 ranking was based on the performance measure H, DataFlow machines would outperform ControlFlow machines. This statement is backed up with performance data presented in the next section.

Most Recent Performance Data

A survey of recent implementations of various algorithms using the DataFlow paradigm can be found in [7]. Future trends in the development of the DataFlow paradigm can be found in [8]. For comparison purposes, future trends in the ControlFlow paradigm can be found in [9].

Some of recent implementations of various DataFlow algorithms interesting within the context of the performance measure H are summarized below.

(1) Lindtjorn et al. [10], proved: (T) A speed-up of about 70 over a CPU and 14 over a GPU machine (application: Schlumberger, GeoPhysics), (P) Using a 150MHz DataFlow machine, and (A) Packaged as 1U.

The algorithm involved was Reverse Time Migration (RTM).

Starting from Acoustic wave equation:

$$\frac{\partial^2 u}{\partial t^2} = \mathrm{v}^2 \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right)$$

where u is pressure and v is velocity

(2) Oriato et al. [11], proved: (T) That two DataFlow nodes outperform 1,900 CPU cores (application: ENI, The velocity-stress form of the elastic wave equation), (P) Using sixteen 150MHz FPGAs, and (A) Packaged as 2U.

The algorithm involved (3D Finite Difference) was:

$$\frac{\partial v_i(\vec{x},t)}{\partial t} - b(\vec{x}) \frac{\partial \sigma_{ij}(\vec{x},t)}{\partial x_j} = b(\vec{x}) \left[f_i(\vec{x},t) + \frac{\partial m^a_{ij}(\vec{x},t)}{\partial x_j} \right],$$

$$\frac{\partial \sigma ij(\vec{x},t)}{\partial t} - \lambda(\vec{x}) \frac{\partial v_k(\vec{x},t)}{\partial x_k} \delta_{ij} - \mu(\vec{x}) \left[\frac{\partial v_i(\vec{x},t)}{\partial x_j} + \frac{\partial v_j(\vec{x},t)}{\partial x_i} \right] = \frac{\partial m^s_{ij}(\vec{x},t)}{\partial t}.$$

(3) Mencer et al. [12] proved: (T) That two DataFlow nodes outperform 382 CPU cores (application: ENI, CRS 4 Lab, Meteorological Modelling), (P) Using a 150MHz DataFlow machine, and (A) Packaged as 1U.

The algorithm involved (Continuity (1) and (2) and thermal energy (3) equations) was:

$$\frac{\partial p_s}{\partial t} = -\int_0^1 \nabla \cdot (\vec{V}_h \frac{\partial p}{\partial \sigma}) d\sigma \qquad (1)$$

$$\frac{\partial q}{\partial t} = -\frac{u}{ah_x} \frac{\partial q}{\partial \lambda} - \frac{v}{a} \frac{\partial q}{\partial \varphi} - \dot{\sigma} \frac{\partial q}{\partial \sigma} + F_q. \qquad (2)$$

$$\frac{\partial\theta}{\partial t} = -\frac{u}{ah_x}\frac{\partial\theta}{\partial\lambda} - \frac{v}{a}\frac{\partial\theta}{\partial\varphi} - \dot{\sigma}\frac{\partial\theta}{\partial\sigma} + F_\theta. \qquad (3)$$

(4) Stojanović et al. [13] proved: (T) Speed-up of about 10, (P) Power reduction of about 17 , and (A) All that on an 1U card, comparing it with i7 CPU.

The algorithm involved (Gross Pitaevskii equation) was:

$$i\hbar\frac{\partial}{\partial t}\Phi(\mathbf{r},t) = \left(-\frac{\hbar^2\nabla^2}{2m} + V_{\text{ext}}(\mathbf{r}) + g|\Phi(\mathbf{r},t)|^2\right)\Phi(\mathbf{r},t)$$

(5) Chow et al. [14] proved: (T) Speed-up of about 163, (P) Power reduction of about 170, and (A) All that on an 1U card, comparing it with quad core CPU.

The algorithm involved (Monte Carlo simulation) was:

$$I \approx \langle f_H \rangle_N = \frac{1}{N}\sum_{i=1}^{N} f_H(\vec{x}_i)$$

where $\vec{x}_i$ is the input vector, N is the number of sample points, and $\langle f_H \rangle_N$ is the sampled mean value of the quantity.

(6) Arram et al. [15] proved: (T) Speed-up of about 13 over a 20 core CPU and 4 over a GPU machine, (P) Using one 150MHz DataFlow machine, and (A) Packaged as 1U.

The algorithm involved (Genetic Sequence Alignment) was based on FM-index. This index combines the properties of suffix array (SA) with the Burrows-Wheeler transform (BWT).

SA interval is updated for each character in pattern Q, moving from the last character to the first:

$$k_{new} = c(x) + s(x, k_{current} - 1)$$
$$l_{new} = c(x) + s(x, l_{current}) - 1$$

where pointers k and l are respectively the smallest and largest indices in the SA which starts with Q, c(x) (frequency) is the number of symbols in the BWT sequence that are lexicographically smaller than x and s(x, i) (occurrence) is the number of occurrences of the symbol x in the BWT sequence from the 0th position to the ith position.

(7) Guo et al. [16] proved: (T) Speed-up of about 517 over a CPU and 28 over a GPU machine, (P) Using one 150MHz DataFlow machine, and (A) Packaged as 1U.

The algorithm involved (Gaussian Mixture Models) was:

$$p(\mathbf{x}|\lambda) = \sum_{i=1}^{M} w_i \, g(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

where x is a D-dimensional continuous-valued data vector (i.e. measurement or features), w_i, i = 1, …, M, are the mixture weights, and $g(x|\mu_i, \Sigma_i)$, i = 1, …, M, are the component Gaussian densities.

Conclusion

This viewpoint sheds more light on the programming paradigm and the most recent performance achievements of the DataFlow concept (more details can be found in [8, 17]). It can be of interest to those who have to make decisions about future developments of their BigData centers. It opens up a new important problem: The need for a development of a public cloud of ready-to-use applications.

Some researches compare the use of MultiCore architectures to plowing with X horses and the use of ManyCore architectures to plowing with 1,000X chickens; we compare the use of DataFlow architectures to plowing with 1,000,000X ants ☺.

One question is what method of plowing is the fastest (computational speed)? If one plows a horizontal field (i.e., if one is using relatively simple benchmarks, like Linpack), horses are the fastest. If one plows a steep field (i.e., if one uses the benchmarks that favorize ManyCore over MultiCore), chickens are the fastest. However, if one plows a vertical stone field (BigData), ants are the fastest, and are the only animals that can do the task.

Another question is the cost of feeding the animals used for plowing (power consumption)? A hay stack for horses is expensive, as well as multiple corn bit boxes for chickens. However, one can feed ants with crumbs left over from the plow driver breakfast (power consumption is negligible, which enables that seismic data processing is completed even on the seismic vessels, i.e., at the very source of data, without moving data to remote mainland data centers – data movement is expensive [2]).

Still another question is where to keep the animals (equipment size)? For horses one needs big stable, for chickens one needs small but multiple chicken cages. However, with ants there is no need for a new and expensive data center with “bricks and mortar”. They easily find a way to live in a corner crack of an existing data center ☺.

REFERENCES

[1] Flynn, M., Mencer, O., Milutinovic, V., Rakocevic, G., Stenstrom, P., Trobec. R., Valero, M., "Moving from Petaflops to Petadata," *Communications of the ACM*, ACM, New York, NY, USA, Volume 56, Issue 5, May 2013, pp. 39-42.

[2] Dennis, J., Misunas, D., "A Preliminary Architecture for a Basic Data-Flow Processor," *Proceedings of the ISCA '75 the 2nd Annual Symposium on Computer Architecture*, *ACM*, New York, NY, USA, 1975, pp. 126-132.

[3] Agerwala, T., Arvind, -., "Data Flow Systems: Guest Editors' Introduction," *IEEE Computer*, Feb. 1982, Vol. 15, No. 2, pp. 10-13.

[4] Mencer, O., "Multiscale Dataflow Computing," *Proceedings of the Final Conference Supercomputacion y eCiencia, SyeC,* Barcelona, Catalonia, Spain, May 27 - May 28, 2013.

[5] Resch, M., "Future Strategies for Supercomputing," *Proceedings of the Final Conference Supercomputacion y eCiencia*, *SyeC,* Barcelona, Catalonia, Spain, May 27 - May 28, 2013.

[6] Flynn, M., "Area - Time - Power and Design Effort: The Basic Tradeoffs in Application Specific Systems," *Proceedings of the 2005 IEEE International Conference on Application-Specific Systems and Architecture Processors (ASAP'05)*, Samos, Greece, July 23-July 25, 2005.

[7] Salom, J., Fujii, H., "Overview of Acceleration Results of Maxeler FPGA Machines," *IPSI Transactions on Internet Research,* July 2013, Volume 5, Number 1, pp. 1-4.

[8] -, -., *Maxeler Technologies FrontPage*, http://www.maxeler.com/content/frontpage/, London, UK, October 20, 2011.

[9] Patt, Y., "Future Microprocessors: What Must We do Differently if We Are to Effectively Utilize Multi-core and Many-core Chips?," *IPSI Transactions on Internet Research,* January 2009, Volume 5, Number 1, pp. 1-9.

[10] Lindtjorn, O., Clapp, G., Pell, O., Mencer, O., Flynn, M., Fu, H., "Beyond Traditional Microprocessors for Geoscience High-Performance Computing Applications," *IEEE Micro*, Washington, USA, March/April 2011, Vol. 31, No. 2, pp. 1-9.

[11] Oriato, D., Pell, O., Andreoletti, C., Bienati, N., „FD Modeling Beyond 70Hz with FPGA Acceleration," *Maxeler Summary, http://www.maxeler.com/media/documents/MaxelerSummaryFDModelingBeyond70Hz.pdf,* Summary of a talk presented at the SEG 2010 HPC Workshop, Denver, Colorado, USA, October 2010.

[12] Mencer, O., "Acceleration of a Meteorological Limited Area Model with Dataflow Engines," *Proceedings of the Final Conference Supercomputacion y eCiencia*, *SyeC,* Barcelona, Catalonia, Spain, May 27 - May 28, 2013.

[13] Stojanovic, S. et al, "One Implementation of the Gross Pitaevskii Algorithm," *Proceedings of Maxeler@SANU, Belgrade,* Serbia, April 8, 2013.

[14] Chow, G. C. T., Tse, A. H. T., Jin, O., Luk, W., Leong, P. H. W., Thomas, D. B., "A Mixed Precision Monte Carlo Methodology for Reconfigurable Accelerator Systems," *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA),* Monterey, CA, USA, February 2012, pp. 57-66

[15] Arram, J., Tsoi, K. H., Luk, W., Jiang, P., "Hardware Acceleration of Genetic Sequence Alignment," *Proceedings of 9th International Symposium ARC 2013*, Los Angeles, CA, USA, March 25-27, 2013. pp. 13-24.

[16] Guo, C., Fu, H., Luk, W., "A Fully-Pipelined Expectation-Maximization Engine for Gaussian Mixture Models," *Proceedings of 2012 International Conference on Field-Programmable Technology (FPT)*, Seoul, S. Korea, 10-12 Dec. 2012, pp. 182 - 189

[17] Flynn, M., Mencer, O., Greenspon, I., Milutinovic, V., Stojanovic, S., Sustran, Z., "The Current Challenges in DataFlow Supercomputer Programming," *Proceedings of ISCA 2013*, Tell Aviv, Israel, June 2013.