Opportunities and choice in a new vector era

A Nowak[1]
[1] CERN openlab CTO Office, Geneva, Switzerland

E-mail: Andrzej.Nowak@cern.ch

Abstract. This work discusses the significant changes in computing landscape related to the progression of Moore's Law, and the implications on scientific computing. Particular attention is devoted to the High Energy Physics domain (HEP), which has always made good use of threading, but levels of parallelism closer to the hardware were often left underutilized. Findings of the CERN openlab Platform Competence Center are reported in the context of expanding "performance dimensions", and especially the resurgence of vectors. These suggest that data oriented designs are feasible in HEP and have considerable potential for performance improvements on multiple levels, but will rarely trump algorithmic enhancements. Finally, an analysis of upcoming hardware and software technologies identifies heterogeneity as a major challenge for software, which will require more emphasis on scalable, efficient design.

1. Introduction

CERN openlab and its Platform Competence Center spent the past decade investigating mainstream and upcoming platform technologies. During that time, the computing landscape has witnessed many changes, but two are of particular importance. On one end of this decade, intrinsic improvements in single-thread processor efficiency ceased to be sourced from manufacturing process improvements and aggressive frequency scaling, and had to be provided through microarchitectural optimizations such as out of order enhancements, changes in branch predictors, cache optimizations, buffer size increases and so on. High Energy Physics (HEP) software traditionally makes pretty good use of such mechanisms. On the other end of this decade, we find ourselves snowed under with vectors, hardware threads, cores – all various forms of parallelism.

This new era presents challenges as it presents opportunities. The discussion contained in the paper presented addresses these main points.

2. Observations

2.1. Hardware

Worries about computing platform performance often stem from an inefficient use of available resources - which is a generic and persistent challenge. One such challenge in the past was the growing compute-memory gap, illustrated on the classic Hennessy & Patterson plot (Figure 1).

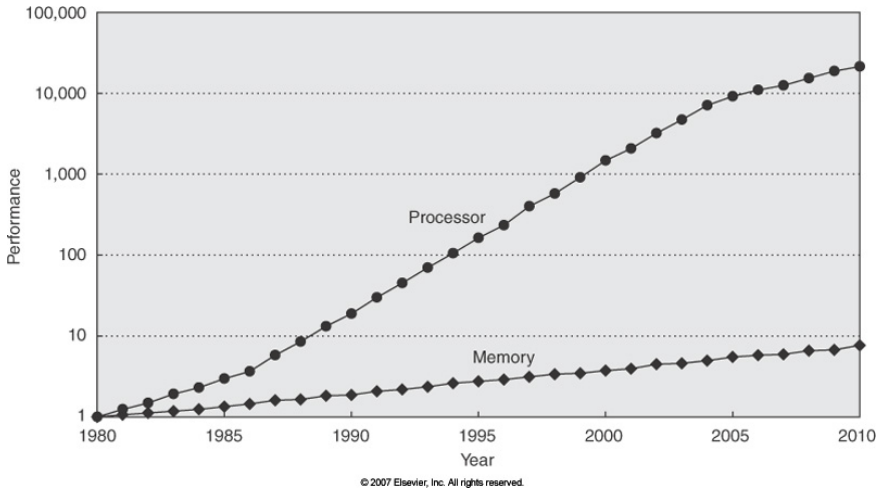

Figure 1: CPU-memory performance gap. Modelled after ”Computer Architecture”: Hennessy, John L.; Patterson, David A.

In modern computing, it is less common for bottlenecks in a platform to shift in a major way, e.g. from disk to memory bandwidth, and thus even well-tuned workloads are rarely adversely affected by hardware upgrades. A notable exception is the rise of accelerators, which in a “step function” offered difficult to extract compute capabilities, but plenty of bandwidth to go along (Figure 2).

Let us consider the recent history of mainstream compute capabilities. Vector width has quadrupled on x86 since the introduction of MMX (and went much further on GPUs!), hardware threading made a triumphant comeback after a brief appearance in the Pentium 4 family, and the “Haswell” microarchitecture from Intel now features 8 execution ports in lieu of 6 on earlier models. Core count in a chip has grown from 2 in awkward MCP designs to over 60 in the Intel KNC, while multi-socket, multi-node installations are standards in datacenters. All in all, the appearance of a vast land of opportunity created by the simultaneous expansion of multiple performance dimensions has a strong impact on the ratio between the theoretical limits of computing efficiency and the targets actually achieved – as discussed further.

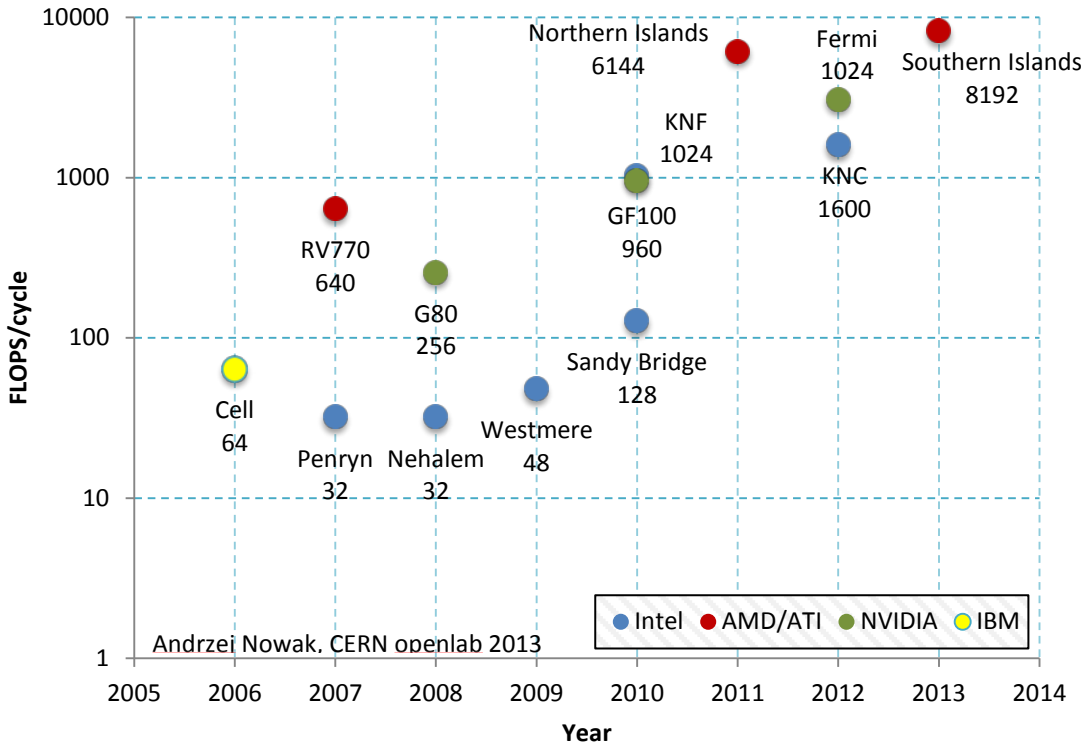

Figure 2: Peak single precision, general purpose FLOPs/cycle in a single chip

This illustrates the need for a shift in programming and optimization focus, in particular in compute-oriented workloads - from generic latency and hotspot-oriented efforts to sweeping redesigns of whole software packages. While it is common to extrapolate Moore's Law and its benefits at least 5 years into the future, industry specialists warn of major movements in connectivity, memory and energy efficiency. For example, the ratio of bytes per FLOP is no longer a compute density metric, but also one that implies an optimal or suboptimal energy balance in a chip.

Table 1: Memory operations and their energy costs in a platform

Operation	Energy cost
L1 access	2 pJ
L2 access	150 pJ
RAM	2 000 pJ
FLOP	Future - ?

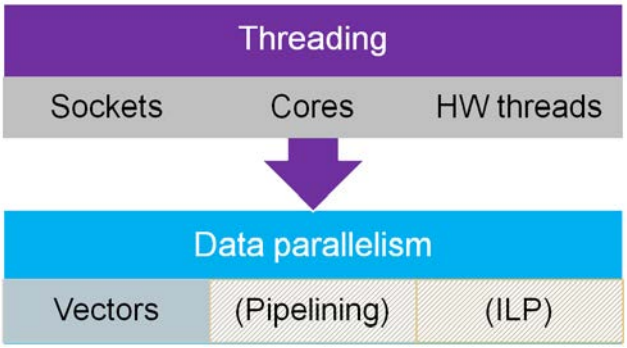

Figure 3: Hardware parallelism mapped onto software

This trend can be noted by observing the widespread community interest in accelerators, vectors and even new architectures – after many years of indivisible x86 reign on our desktops and in our datacenters, numerous experiments with ARM, NVIDIA and even MIPS architectures are underway.

2.2. Software stagnation

On the other side, production HEP software remained the same from the point of view of its capability to take advantage of technical novelties. As discussed in [1], large, code-heavy C++ based

frameworks dominate, with two major toolkits, ROOT [2] and Geant4 [3,4], still single-threaded and lacking vectorization (for the most part). It should not come as a surprise that, roughly speaking, the gap in performance between the theoretical limit and optimized code is 10x, and the distance between the performance of scalar, single-threaded HEP code and optimized code is another 10x. These numbers are decomposed into more detailed factors in Table 2 and Table 3, and explained in detail in [1].

Table 2: Multiplicative factors for parallelism in three different software and datacentre scenarios

	SIMD	ILP	HW THREADS	CORES	SOCKETS
THEORY	4	4	1.35	8	4
OPTIMIZED	2.5	1.43	1.25	8	2
HEP	1	0.80	1	6	2

Table 3: Multiplied factors for parallelism in three different software and datacentre scenarios

	SIMD	ILP	HW THREADS	CORES	SOCKETS
THEORY	4	16	21.6	172.8	691.2
OPTIMIZED	2.5	3.57	4.46	35.71	71.43
HEP	1	0.80	0.80	4.80	9.60

We need to program for tomorrow's hardware – today.

The important lesson here touches not only on modularity and adaptability to changes in hardware, both of which must become the daily bread of many key developers, but also on the viabililty, sustainability and performance of the software engineering process in HEP. New challenges linked to heterogeneity, discussed further, will put our methods to an even more stringent test. In 2007, when the many-core era was just around the corner, CERN openlab began its efforts of community evangelization in preparation for the inevitable explosion in datacentre core count. It quickly became apparent that our software would benefit from more forms of hardware parallelism than one, which fortunately still map onto a manageable set on the software side (Figure 3). However, many physicists we spoke to did not have the bandwidth to worry about issues such as hardware underutilization. Now the gap has grown sufficiently large for many to find the resources to address this challenge. For its part, CERN has initiated the Forum on Concurrent Programming Models and Frameworks and is investing in major improvements in Geant4, such as Geant-V and the multi-threaded Geant4 prototype, and ROOT. Another prominent effort is the EU FP7 "ICE-DIP" project, which aims to develop next-generation data acquisition capabilities using new technologies.

Large HEP jobs are deeply affected by object-oriented development, to the extent that top functions in a flat profile consume low single-digit percentages of time or instructions, while it takes hundreds if not thousands of entries to account for the bottom 50% of the profile – which means that even if we get rid of them completely (hardly a possibility), the workload would run at most twice as fast. Such frameworks are almost beyond hope when it comes to the *efficient* use of opportunities such

as vectorization, and need a data oriented design approach to benefit from them. It's not all bad, however: prominent examples of success exist as well. For example, the GSI team working on the CBM/ALICE online codes achieved a 120'000x speedup on their code. The interesting part is that only a factor of 120x came from microarchitectural optimizations and parallelization, while a factor of 1000x came from implementing a significantly more optimal algorithm (Figure 4). There is also a good result from the CERN openlab side: work done on the MLFit benchmark allowed it to scale by nearly 2x when moving from an Intel "Westmere" based server to a "Sandy Bridge" one (Figure 5).

	Stage	Description	Time/track	Speedup
		Initial scalar version	12 ms	–
Intel	1	Approximation of the magnetic field	240 μs	50
Intel	2	Optimization of the algorithm	7.2 μs	35
Intel	3	Vectorization	1.6 μs	4.5
Cell	4	Porting to SPE	1.1 μs	1.5
Cell	5	Parallelization on 16 SPEs	0.1 μs	10
		Final simdized version	0.1 μs	120000

10000x faster on any PC

Comp. Phys. Comm. 178 (2008) 374-383

Figure 4: Speedups for optimized ALICE/CBM online code, figure copied from I. Kisel et al. [5][6]

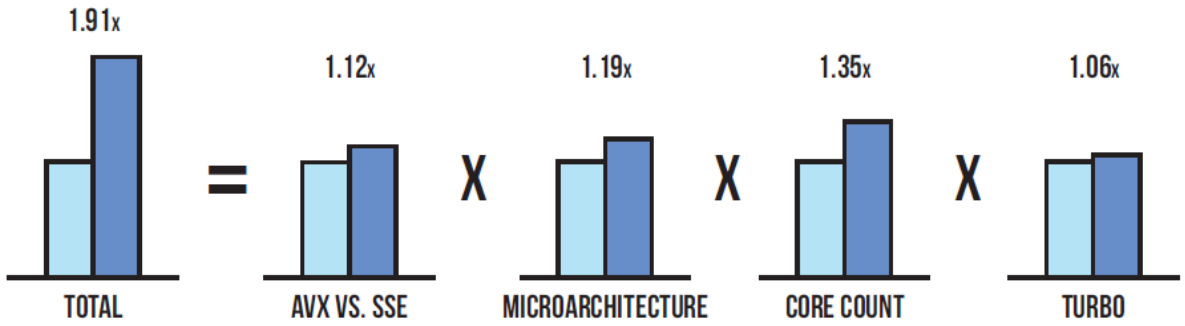

Figure 5: "Westmere" (light blue) vs. "Sandy Bridge" (dark blue) server performance, frequency scaled

In what concerns specific technologies for, say, vectorization, they are differentiated not just by their features, but also by the syntax or notation they employ and their implementation (library, compiler extension, etc). While new syntax options such as Cilk+ are quite appealing, to the extent that they're desired in the C++ standard, there has yet to appear a sufficiently convincing technical implementation that provides convenience and performance at the same time. On the multi-threading front, TBB is being integrated into a prototype version of Geant4, also with notable success.

There is, of course, no guarantee that vector growth will continue, but it is still a growing trend, with AVX3 sporting 512-bit vectors on mainstream Xeon systems, and much wider vector-like units still being deployed in GPUs en-masse.

Finally, even though the case for parallelism on multiple levels might be clear by now, it will still take some work to establish which is the best path to pursue, which technologies are the most promising (and sustainable) ones, and which syntax, notation and programming concept will fit the HEP community.

3. Heterogeneity

One emerging challenge with respect to contemporary computing strategies and software development is heterogeneity. It has been around for a long time at the datacentre level, where for example a Grid user only sees a virtualized interface to his system and does not know which

capabilities the CPU assigned to his job will have. At the chip level, AMD has its “Trinity” APU technology, while Intel introduced graphics processors supporting “Sandy Bridge” processors and its successors. Very few (experimental) software packages make use of such hardware. Finally, at the platform level, we see heterogeneity not only at the level of compute capabilities of interconnected devices but also in terms of their architecture, memory configuration, integration and so on. Before such a device is engaged to collaborate with the main CPU, many questions must be asked, also at the software level. How will the device feed on data and communicate? Will it have sufficient memory, performing as expected? Will it meet latency constraints? Will it provide floating point results compatible with the main processor? Should the code run natively on the accelerator or be offloaded when the time is right? The answer to those questions and actions taken will depend on the problem and the structure of next-generation software frameworks. Another platform related question is that of memory – in recent years there has been a trend of stratification. Two levels of caches expanded to three and multi-socket systems provide multiple NUMA domains. A hot topic in this domain is high bandwidth, low latency, potentially non-volatile memory, which could act as a layer between L3 and RAM, or play a role in storage (Figure 6 and Table 4).

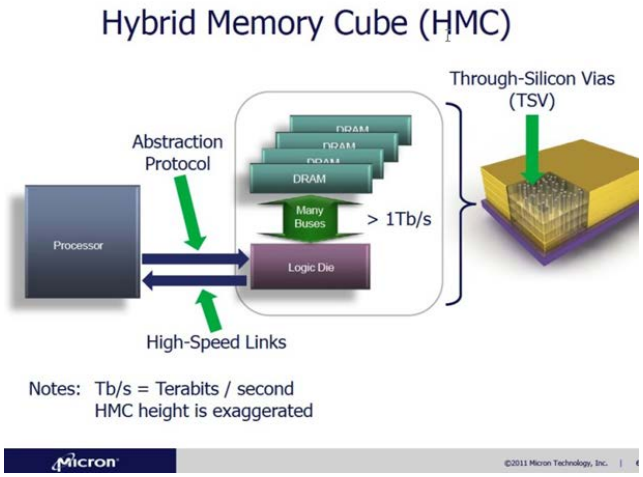

Figure 6: Micron HMC (from S. Borkar, Intel)

	Bandwidth	**Energy**
DDR3 (Today)	> 10GB/s	> 50 pJ / bit
HMC (Prototype)	> 100 GB/s	< 10 pJ / bit

Table 4: HMC prototype parameters

The software side will feel pressure on two fronts. On one side, compilers and toolsets might begin to diverge, and the OS kernel will have to efficiently arbitrate communication between the host and slot-in devices. Such has been the experience of CERN openlab while working on the Intel MIC project since 2008, where none of the physics benchmarks used or their dependencies were fully ready for cross-platform compilation and usage. The porting times shown in Table 4 illustrate that a small benchmark such as the Trackfitter was easy to port, even though it required a manual reimplementation of several key functions, to match the architecture of the KNF. MLFit, which was written using large standards (OpenMP and MPI) and stripped of ROOT dependencies, was essentially a matter of recompilation and updating the build system. However, the first port of the multi-threaded Geant4 prototype, which relies on a lot of Geant4 source code and has several external dependencies, took more than a month to get right. The column showing the time needed for new (updated) ports demonstrates that in our experience, introducing support for a certain degree of heterogeneity was an investment that paid off and did not require major reviews.

	LOC	1st port time	New ports	Tuning
TF	< 1’000	days	N/A	2 weeks
MLFit	3’000	< 1 day	< 1 day	weeks
MTG4	2’000’000	1 month	< 1 day	< 1 week

Table 5: Approximate KNF/“Larrabee” (MIC) porting and tuning times by HEP benchmark

The low-power ARM architecture is making its 64-bit entry into datacenters worldwide, and those in the HEP community who wish to use it [7][8] will face challenges very similar to those experienced in our work with Intel MIC. For example, compiler support for NEON vector instructions is still quite young, and several other areas in popular Open Source software need attention. As recently as in 2012, it was still a time consuming feat to properly install and support Linux on one of the “desktop” ARM development boards.

On the other software front, it is not yet apparent how to maintain a single source that could be compiled to or offloaded to different backends [9], and whether it should be maintained as “straight” C/C++ code or as targeted kernels, or perhaps even in another form. Several projects tried to bridge this productivity and engineering gap – RapidMind, Intel’s Ct, OpenCL, custom Intel compiler extensions and several others – but none of them have managed to provide a portable source AND portable performance at the same time. Some established industry standards come to the rescue, e.g. OpenMP4 now includes accelerator and offload directives that were pre-published with OpenACC, but it will still take some time before the industry comes to a well-defined conclusion.

A particular and painful consequence of symmetric processing is the challenge of numerical convergence [10][11]. In some cases, there is no other choice but to accept that floating point results will diverge between the CPU and an accelerator or co-processor. Vectors might be processed differently, data and operation ordering may change, associativity may vary, compilers will generate different code and “fast” function variants might be used as defaults. In any modern “offload” device (GPUs and Intel MIC included) there exist two or three variants of math functions – those that strictly adhere to standards but are excruciatingly slow, those that explicitly sacrifice accuracy for speed, and those that offer a reasonable compromise by operating on limited ranges of input arguments. Unfortunately for coders, these new options often imply a revision of floating-point in the source – an undertaking that is not concluded overnight on frameworks with 5 million lines of code. Ultimately, such an exercise can be a very useful one, in particular to develop methods that assess the quality of computation results through means other than their bit-for-bit binary compatibility on multiple platforms. That is also why it is important to closely follow efforts on math library optimization, such as [12].

4. Conclusion

As always, the computing landscape is changing rapidly. However, it has become the responsibility of the platform and system user to “seize the day”, and to turn all the hardware in a computing platform into an advantage – as opposed to neglecting silicon that has already been paid for. This should be taken into account in the light of general optimization truths which suggest that optimization should start with the algorithm, as that often is where the highest potential for gains resides.

At least now, it would seem that next generation code must make some choices between programmability and performance – neither can be sacrificed and neither can be ignored. Universal, open standards must continue to play a leading role for reasons of sustainability and interoperability. Scaling in multiple hardware and software dimensions seems indispensable – that includes numerous aspects such as parallelization, vectorization, multi-process domain, message passing readiness,

"asynchronicity" and lack of coherency. Well-designed algorithms and frameworks will be a major contributor to success in this area. Finally, power efficiency is becoming a practical worry as well – future chips might require an optimal or specific balance of data movement and computation.

There is a long road ahead of the HEP community, one that will surely take many turns and hold many surprises. Without doubt, many mainstream technologies the community uses today have already peaked, and it could well be that we currently find ourselves at a crossroads similar to the one that took us into the PC era [13].

5. Acknowledgements

The dissemination of this work was supported by "ICE-DIP", an EU FP7 Marie Curie grant, #316596.

6. References

[1] Jarp S, Lazzaro A and Nowak A 2012 The future of commodity computing and many-core versus the interests of HEP software *J. Phys. Conf. Ser.* **396** 052058

[2] Brun R and Rademakers F 1997 ROOT — An object oriented data analysis framework *Nucl. Instruments Methods Phys. Res. Sect. Accel. Spectrometers Detect. Assoc. Equip.* **389** 81–6

[3] Agostinelli S, Allison J, Amako K, Apostolakis J, Araujo H, Arce P, Asai M, Axen D, Banerjee S, Barrand G, Behner F, Bellagamba L, Boudreau J, Broglia L, Brunengo A, Burkhardt H, Chauvie S, Chuma J, Chytracek R, Cooperman G, Cosmo G, Degtyarenko P, Dell'Acqua A, Depaola G, Dietrich D, Enami R, Feliciello A, Ferguson C, Fesefeldt H, Folger G, Foppiano F, Forti A, Garelli S, Giani S, Giannitrapani R, Gibin D, Gómez Cadenas J J, González I, Gracia Abril G, Greeniaus G, Greiner W, Grichine V, Grossheim A, Guatelli S, Gumplinger P, Hamatsu R, Hashimoto K, Hasui H, Heikkinen A, Howard A, Ivanchenko V, Johnson A, Jones F W, Kallenbach J, Kanaya N, Kawabata M, Kawabata Y, Kawaguti M, Kelner S, Kent P, Kimura A, Kodama T, Kokoulin R, Kossov M, Kurashige H, Lamanna E, Lampén T, Lara V, Lefebure V, Lei F, Liendl M, Lockman W, Longo F, Magni S, Maire M, Medernach E, Minamimoto K, Mora de Freitas P, Morita Y, Murakami K, Nagamatu M, Nartallo R, Nieminen P, Nishimura T, Ohtsubo K, Okamura M, O'Neale S, Oohata Y, Paech K, Perl J, Pfeiffer A, Pia M G, Ranjard F, Rybin A, Sadilov S, Di Salvo E, Santin G, Sasaki T, et al 2003 Geant4—a simulation toolkit *Nucl. Instruments Methods Phys. Res. Sect. Accel. Spectrometers Detect. Assoc. Equip.* **506** 250–303

[4] Allison J, Amako K, Apostolakis J, Araujo H, Dubois P A, Asai M, Barrand G, Capra R, Chauvie S and Chytracek R 2006 Geant4 developments and applications *Nucl. Sci. Ieee Trans.* **53** 270–8

[5] Kisel I, Kulakov I and Zyzak M Parallel Implementation of the KFParticle Vertexing Package for the CBM and ALICE Experiments *Computing in High Energy and Nuclear Physics 2012*

[6] Kisel I, Kulakov I and Zyzak M Parallel Algorithms for Track Reconstruction in the CBM Experiment Computing in High Energy and Nuclear Physics 2012

[7] Neufeld N Measurements of the LHCb software stack on the ARM architecture

[8] Elmer P Explorations of the viability of ARM and Intel Xeon Phi for Physics Processing

[9] Rohr D, Gorbunov S, Szostak A, Kretz M, Kollegger T, Breitner T and Alt T 2012 ALICE HLT TPC Tracking of Pb-Pb Events on GPUs *J. Phys. Conf. Ser.* **396** 012044

[10] Corden M J Differences in floating-point arithmetic between Intel Xeon processors and the Intel Xeon Phi coprocessor

[11] Corden M J and Kreitzer D 2009 *Consistency of floating-point results using the intel compiler or why doesn't my application always give the same answer* (Technical report, Intel Corporation, Software Solutions Group)

[12] De Dinechin F From CRLibm to Metalibm: assisting the production of high-performance proven floating-point code

[13] Jarp S, Simmins A, Yaari R and Tang H 1995 *PC as physics computer for LHC?*