

Defining the foundations of Programmability Research

Tim Mattson, Intel, Microprocessor Technology laboratories, DuPont Washington

For the last 25 years, the HPC community has been working to find the right programming technology to make the “general purpose programmer” a parallel programmer. The result is literally hundreds of parallel programming languages, APIs and high level tools. And while MPI and OpenMP have had limited success, for the most part all this work has failed to significantly increase the fraction of programmers who consider themselves to be parallel programmers.

Now with multi-core processors, there is a renewed push to find that magic set of technologies that will make parallel programming a skill routinely expected of any professional programmer. New languages are popping up left and right. HPCS languages, data parallel languages, frameworks to make parallel programming easy, etc.

But before we start spinning off new parallel programming technologies, maybe we should pause and ask ourselves why we expect to succeed this time when so many before us have failed. If we just create new parallel programming technologies without learning from the past, won't we just repeat the mistakes of old? In my view, we made a number of fundamental errors over the last 25 years. And so far as we move into the multi-core era, we are just making them all over again.

First and foremost, we have to stop thinking like engineers and start thinking like scientists. Engineers come up with great ideas, build something based on the ideas, show they work, declare victory, and then move on. Scientists develop hypothesis, design experiments to test them, run the experiments, submit the results to peer review, and then refine their ideas to build theories. The scientific method applied to the development of parallel programming technologies, something totally missing from much of the research in parallel programming over the last 25 years, is the only way to get it right this time around.

For the scientific method to work, we must address two foundational issues. First we must define a common terminology, i.e. a *human-language* for parallel programming. This is the only way to move beyond “marketing speak”, beyond empty if not childish statements of “my language is better than yours” to systematic and productive discourse exposing the strengths and weaknesses of different approaches. Second, we need to define the raw materials of an experimental methodology for programmability research; i.e. a set of standard programmability benchmarks.

To be productive at all, our language of programmability and the set of benchmarks must be defined by a dedicated community of parallel programming researchers. Hence, I can't pretend to have the answers. But I do have proposals to start a productive dialog on these issues.

For a human languages of programmability, I propose that we adopt Thomas Green's cognitive dimensions of information notations. These define a discussion framework to make more explicit the qualitative and often subjective features of an information system. The full set of dimensions includes:

- Viscosity: How hard is it to introduce a small change in a program once the program has been written.
- Hidden dependencies: Does the programming notation let the programmer make changes in one part of a program that impact another part of the program without visible indicators of this connection in the source code.
- Abstraction gradient: how much effort does the programmer have to expend building abstractions in order to make use of the notation.
- Abstraction depth: Does the language allow the programmer to build his or her own abstractions to support the programming process.

There are around a dozen of these dimensions. We will need to add a few more to meet the specific needs of parallel programming, but to whatever extent possible, we should try to stick to the original set.

If we as a community explore Green's dimensions and modify them to "make them our own", we will have the human-language we need to support a systematic dialog about programmability. Note that applying Green's techniques to programming languages is not a new idea. They were initially developed to analyze visual programming languages. More recently, Steven Clarke of Microsoft Research in Cambridge used them to study the design of C#.

With a language of programmability in place, we can conduct and discuss experiments in programmability. This implies that we need a common set of programmability benchmarks. A high quality set of benchmarks should have the following characteristics:

- They should contain synthetic applications that expose the complexity inherent to real application programs; not simple kernels or "toy-codes".
- They should be long enough to be non-trivial, but not so complex that building and maintaining the components of the benchmark suite becomes a major task in and of itself. The goal is research on parallel programming, not the development of the benchmark applications themselves.
- The source code must be simple enough and short enough so people can compare and contrast implementations using different technologies with reasonable cognitive effort.
- They should be defined "on paper" to support development with the widest range of notations, but serial reference implementations in a common language should also be provided.
- The number of components in the benchmark suite should be large enough to cover the most important classes of applications, but not so large as to dissuade researchers from studying the full set.

Past efforts along these lines are the compact synthetic applications from the HPCS program, the Salishan problems from the mid-90's, and the SPLASH benchmarks. Good

starting points for the benchmark suites are the 13 Dwarves from UC Berkeley or the many core benchmarks underdevelopment in Professor Kai Li's group at Princeton.

I am actively looking for collaborators in Industry and Academia to address these issues and lay the foundations for a systematic, science driven methodology to solve the parallel programming problem once and for all. The need for this work is urgent. In fact, until we complete this foundational work, research on parallel programming notations is premature and no more likely to succeed than the hundreds of projects from the 90s

References:

[Clarke2001] Steven Clarke, "Evaluating a new programming language", Proceedings of the 13th Workshop of the Psychology of Programming Interest Group, Bournemouth UK, G. Kadoda (Ed.), pp. 275-289, 2001.

[Green96] T.R.G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: a 'cognitive dimensions' framework", J. Visual Languages and Computing, vol.7 pp. 131-174, 1996.