Performance, Portability, and Dreams

William Gropp
wgropp.cs.illinois.edu
Why Performance Portability?

- HPC is Performance
- A big part of the programming crisis is caused by the challenge of obtaining performance (even) on a single platform
  - This is an unsolved problem
    - Easy example: Implement a barrier. Communicates a single bit of information.
      - Easy to write a simple implementation (e.g., use an atomic counter).
      - Efficient implementations require clever algorithms, attention to memory hierarchies, special instructions, and are still publishable
        - Recent results such as “Efficient Barrier Implementation on the POWER8 Processor”, HiPC 2015
  - Performance portability related to programming productivity
    - And a harder problem that is getting even harder

Platform-specific
Why Is Performance Portability so Hard

• It’s been hard
  • Predicting performance for a single system is very difficult
    • Systems are complex
    • Behavior has random elements
    • Interactions between parts is hard to predict

• After more than 20 years of stability, processor architectures are diversifying and changing
  • More types of systems – e.g., vectors/streams in GPUs
  • Rapid innovation – new instructions, memory architectures, …
  • Effective* use of these requires someone to adapt to the differences
    • Please make it someone else!

• Even if it is someone else
  • Many costs and risks to maintaining multiple versions
Tradeoffs

• Implicitly, performance portability is intended to reduce the amount of work needed to achieve adequate performance to meet the needs that the computing serves.

• How much (programmer re-) work is acceptable to achieve performance portability?

• What constraints or other limitations are acceptable?
  - Choices of data structure, code complexity, reproducibility, compile time, sensitivity to changes in input data, ...
What is the Problem Statement?

• General case (strong performance portability) – get the fastest solution to the problem on all systems – is far too hard – requires picking model, algorithm, data structure, and implementation
  • Best algorithm/data structure choice often unknown
  • Algorithm may depend on platform
    • Proof – parallel algorithms that trade less synchronization against more work vs. sequential algorithms

• Given a family of data structures and an algorithm, choose the data structure instance and implementation
  • E.g., Array index ordering; Structure of arrays vs array of structures vs structure of arrays of structures; sparse matrix ordering

• Given a data structure and an algorithm, generate “good” code that performs well
  • Problem: choices here are important for performance
  • Problem: Still hard even in simple cases
  • Problem: “Extra” semantics in language – e.g., order of operations for floating point, e.g. in dense matrix-matrix multiply, can limit options even if not intended by programmer
Metrics for Success

• What is success?
• Need to quantify both portability and performance
• Should include impact on productivity
  • A performance portable code that is no longer maintainable or that is too brittle for further development is probably not an improvement
• Not an easy linear function
  • Different users and communities may choose different weights for the metric
**Productivity and Performance**

Collaborative Filtering (Weak scaling, 250 M edges/node)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time per iteration (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi</td>
<td>10</td>
</tr>
<tr>
<td>Combblas</td>
<td>100</td>
</tr>
<tr>
<td>Graphlab</td>
<td>1000</td>
</tr>
<tr>
<td>Socialite</td>
<td>10,000</td>
</tr>
<tr>
<td>Giraph</td>
<td>100,000</td>
</tr>
</tbody>
</table>

**Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets**

Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M. Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey
Dangers in Performance Portability

• One easy way – make all performance mediocre
  • One vendor did this in the ‘80s with their vector hardware, to avoid too large a variability in performance
    • Goal was no performance “surprises”
  • Related – predictable performance – a goal and elegant feature of BSP (Bulk Synchronous Programming)
    • How much opportunity for higher but less predictable performance are you willing to give up for predictable performance? Do your users agree with you?

• Another easy way – claim that it can be reduced to a previously solved problem
  • E.g., Claim the compiler can take care of it
    • This is a fantasy
  • We clearly need a good definition…
One Definition

- An application is performance portable if it achieves a consistent ratio of the actual time to solution to either the best-known or the theoretical best time to solution on each platform with minimal platform specific code required.
  - From http://performanceportability.org/perfport/definition/
  - Note that other definitions are mentioned with different focus and levels of precision

- “Best-known” time to solution is a big loophole
  - For a new system, best known is your own best time
  - If there is only one code, and it runs and there is no theoretical best time, the code is performance portable, regardless of the actual performance
    - That consistent ratio is 1 😊

- See more on one view of performance portability at
  - http://performanceportability.org/
What Is Performance Portability?

• Is it:
  • A code is performance portable if it achieves at least 100-X% of the achievable performance on all platforms

• Do I need to add constraints?
  • with the same algorithm
  • and the same data structures
  • and the same input and output data organization and format
  • and the same build system (e.g., makefile)

• How large can X be for this definition to be useful?

• Is X the same for all platforms?
  • Alternately, is there an absolute performance target, and the code is performance portable if the code meets or exceeds that performance on all platforms of interest?

• Is there a scaling of X based on the cost ($) of the platform?
Defining Performance Portability

- And what about the correctness constraints
  - Is the output strongly or weakly deterministic?
  - Is bitwise identical output required?

- What is the definition of achievable performance?
  - FLOPS?
  - FLOPS and Memory Bandwidth (“roofline”)
    [https://dl.acm.org/citation.cfm?id=1498785](https://dl.acm.org/citation.cfm?id=1498785)
  - FLOPS and Memory Bandwidth and Latency (Execution-Cache-Memory (ECM) model)
    [https://link.springer.com/chapter/10.1007%2F978-3-642-14390-8_64](https://link.springer.com/chapter/10.1007%2F978-3-642-14390-8_64)
  - FLOPS and Memory Bandwidth and Instruction Rate (“Achieving high sustained performance in an unstructured mesh CFD application”)
    [https://dl.acm.org/citation.cfm?id=331600](https://dl.acm.org/citation.cfm?id=331600), 1999
What Is Performance Portability?

• Is it:
  • A code is performance portable if it runs with acceptable performance without any source code change (or architecture-specific directives) on the platforms of interest

• This is squishy. What is
  • Acceptable performance
  • Without any source code change
  • Platforms of interest

• What if I make this *more* squishy
  • A code is performance portable if it runs with acceptable performance with no *onerous* source code or build system changes on *most* of the platforms of interest
Some Performance Portability Questions

• “How much performance would you be willing to give up by replacing the two optimal applications by a single one?”

• How much are you willing to spend to achieve performance portability
  • E.g., if maintaining two codes takes 100 FTE/each and recasting a code in a new system takes 250 FTE, is that acceptable? What if it costs 2500 FTE?

• These ask *quantitative* questions about performance portability

• They also get to the heart of *why* someone might want performance portability
Some Different Approaches to Performance Portability

• Language based
  • Existing languages, possibly with additional information
    • Info from pragmas (e.g., align) or compile flags (assume associative)
  • Extensions, especially for parallelism
    • Directives + runtimes, e.g., OpenMP/OpenCL/OpenACC
    • May also relax constraints, e.g., for operation order, bitwise reproducibility
  • New languages, especially targeted at
    • Specific data structures and operations
    • Specific problem domains

• Library based (define mathematical operators and implement those efficiently)
  • Specific data structure/operations (e.g., DGEMM)
  • Specific operations with families of data structures (e.g., PETSc)
    • This is likely the most practical way to include data-structure and even algorithm choice
    • At the cost of pushing the performance portability problem onto the library developers
Some Different Approaches to Performance Portability

- **Tools based**
  - Recognize that the user can always write poorly-performing code
  - Support programming in finding and fixing performance problems
  - Example: Early vectorizing compilers gave feedback about missed vectorization opportunities; trained programmer to write “better” code

- **Programmer support and solution components**
  - Work with programmer to develop code
  - Source-to-source tools to transform and to generate code under programmer guidance
  - Autotuning to select from families of code
  - Database systems to manage architecture and/or system-specific derivatives

- **Magic**
  - Any sufficiently advanced technology is indistinguishable from magic. (Clarke’s 3rd law)
  - Any sufficiently advanced technology is indistinguishable from a rigged demo.

- **Note these approaches are not orthogonal**
  - Successful performance portability requires many approaches, working together
“Domain-specific” languages

• (First – think abstract data-structure specific, not science domain)
• A possible solution, particularly when mixed with adaptable runtimes
• Exploit composition of software (e.g., work with existing compilers, don’t try to duplicate/replace them)
• Example: mesh handling
  • Standard rules can define mesh
    • Including “new” meshes, such as C-grids
  • Alternate mappings easily applied (e.g., Morton orderings)
  • Careful source-to-source methods can preserve human-readable code
  • In the longer term, debuggers could learn to handle programs built with language composition (they already handle 2 languages – assembly and C/Fortran/…)
• Provides a single “user abstraction” whose implementation may use the composition of hierarchical models
  • Also provides a good way to integrate performance engineering into the application
Let The Compiler Do It

• This is the right answer …
  • If only the compiler *could* do it

• Let’s look at one of the simplest operations for a single core, dense matrix transpose
  • Transpose involves only data motion; no floating point order to respect
  • Only a double loop (fewer options to consider)
A Simple Example: Dense Matrix Transpose

- $\text{do } j=1,n$
  - $\text{do } i=1,n$
    - $b(i,j) = a(j,i)$
  - enddo
- enddo

- No temporal locality (data used once)
- Spatial locality only if $(\text{words/cacheline}) \times n$ fits in cache

- Performance plummets when matrices no longer fit in cache
Blocking for cache helps

- do jj=1,n,stridej
  do ii=1,n,stridei
    do j=jj,min(n,jj+stridej-1)
      do i=ii,min(n,ii+stridei-1)
        b(i,j) = a(j,i)
  
- Good choices of stridei and stridej can improve performance by a factor of 5 or more

- But what are the choices of stridei and stridej?
Results: Blue Waters O1
Results: Blue Waters O3

Simple, unblocked code compiled with O3 – 709MB/s
An Example: Stencil Code from a Real Application

- Stencil for CFD code
- Supports 2D and 3D
- Supports different stencil widths
- Matches computational scientists’ view of the mathematics
Another Version of the Same Code

- This version is 4X as fast as the simpler, easier to read code
- Less general code (subset to stencil, problem dimension)
- Same algorithm, data structure, and operations, but transformed to aid compiler in generating fast (and vectorized) code
Illinois Coding Environment (ICE)

• One pragmatic approach
• Assumptions
  • Fast code requires some expert intervention
  • Can’t all be done at compile time
  • Original code (in standard language) is maintained as reference
  • Can add information about computation to code
• Center for Exascale Simulation of Plasma-Coupled Combustion
  • http://xpacc.illinois.edu

• Approach
  • Annotations provide additional descriptive information
    • Block name, expected loop sizes, etc.
  • Source-to-source transformations used to create code for compiler
    • Exploit tool ecosystem – interface to existing tools
    • Original “Golden Copy” used for development, correctness checks
  • Database used to manage platform-specific versions; detect changes that invalidate transformed versions
Example: Dense Matrix Multiply

**Matrix Multiplication**

```c
#pragma @ICE loop=matmul
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = 0; k < n; k++)
            mC[i][j] += mA[i][k] * mB[k][j];
#pragma @ICE endloop
```

```bash
---
Compilation command before tests
buildcmd: make realclean; make CC={compiler} COPT={params}

search:
  tool: opentuner
  time-limit: 30000
  variants-limit: 1000

buildoptions:
gcc:
  params:{'-O':{'default': 3, 'min': 0, 'max': 3}}

Command call for each test
runcmd: ./mmc

tuning: on
matmul:
  rose_uuic:
    - stripmine+:
      loop: 3
      factor: 2.36
    - stripmine+:
      loop: 2
      factor: 2.48
    - interchange+:
      order: 1,3,0,2,4
      unroll*:
        loop: 5
        factor: 2.24
...```
Performance Results

- Dense matrix-matrix multiply
  - 302,680 total variants
  - Subset evaluated (based on results-so-far)
  - 8.2x speedup over gcc compiler with optimization
  - Small but consistent speedup over icc -O3
- Different parameters can be selected/remembered for each platform
  - Within the constraints of the performance parameters considered
#pragma @ICE loop=stencil
for(i = 1; i < x-1; i++) {
    for(j = 1; j < y-1; j++) {
        for(k = 1; k < z-1; k++) {
        }
    }
}
#pragma @ICE endloop

---

#Built command before compilation
prebuildcmd:

#Compilation command before tests
buildcmd:
    make realclean; make CC={compiler} COPT={params}

buildoptions:
    gcc:
        params:{'-O': {'default': 3, 'min': 0, 'max': 3}}
    icc:
        params:{'-O': {'default': 3, 'min': 0, 'max': 3}}

#Command call for each test
runcmd: ./sten3d 1024 20

tuning: on

stencil:
    rose_uwuc:
        - stripmine+:
            loop: 4
            factor: 16..1024
            type: poweroftwo
        - stripmine+:
            loop: 3
            factor: 16..1024
            type: poweroftwo
        - stripmine+:
            loop: 2
            factor: 16..1024
            type: poweroftwo
        - interchange+:
            order: 0,1,3,5,2,4,6
Performance Results

- 3-D Stencil
  - 11,664 variants
  - Max 12.6 sec
  - Min 3.68 sec
  - Speedup over simple code
    - icc: 1.12x
    - gcc: 1.21x
Other Dangers

• How do we know that the performance portable code is correct?
  • Or even if it will compute the same result as the original code
    • And what is “the same result”?

• It is *not enough* to prove that any code transformations are correct
  • MPICH used to test whether the compiler returned the same result in `a` and `c` for these two statements:
    • `a = joe->array[OFF+b+1];`
    • `c = joe->array[OFF+1+b];`
    • Because one major vendor compiler got this *wrong*.

• And you still need to prove that the hardware implements all of the operations correctly
  • And vectorization is already likely to produce results that are not bitwise identical to the non-vector version (which might depend on how data is aligned at runtime)

• Question: How do you test that the performance portable code is computing what is intended?

• Proving code transformations correct is *necessary* but not *sufficient*
So What Is Performance Portability?

• Rather than define whether a code is (or is not) performance portable, look at the goals
  • Make it easier for end users to run an application code effectively on different systems
    • for some set of systems – not necessarily every possible system
    • May focus on the workflow or the I/O performance, rather than any single code
  • Make it easier for developers to write, tune, and maintain an application for multiple systems
    • Allows a tradeoff between one code and several, based on what’s easier
Summary

• Don’t underestimate the difficulty
  • I don’t believe “strong” performance portability is possible

• Don’t give up
  • There is a lot that can be done to support users and improve performance resilience

• Accept different approaches
  • Different communities, expectations, goals

• Be precise about your goal and accomplishment
  • Let this be a No Hype zone