Productivity with current HPC programming models

Matthias S. Müller, RWTH Aachen University
VI-HPS 10th Anniversary Workshop

June 23, 2017  Seeheim
Motivation and Disclaimer

• Parallel programming is *hard*
• We need to be scientific about solving these problems
• We would all like parallel programming to be easier and more fun, but to accomplish that, we need to focus on the real problems

Comparing two programming models:
First – are you comparing programming *models*, programming *systems*, or *implementations* of programming systems?
  - • Answer – Almost always implementations
  - • Implication – No paper should be accepted that claims to compare X to Y when all it does is compare an implementation of X on Z to an implementation of Y on Z

Source: Bill Gropp “Thinking about parallelism and programming” SC2016
Disclaimer II and more information

- See chapter 10 of Sandra Wienke’s thesis: “Methodology of Development Effort Estimation in HPC”
- However:
  - Solid statistics to make valid statements about real programming model is not available
Some personal opinion about the productivity of programming models
Size of the standard
Evolution of MPI and OpenMP Standard

Size of Standard Specification

Year of publication

Pages


 MPI OpenMP
OpenMP vs MPI

1 : 0
Productivity
Case Study: NINA

- Software\textsuperscript{*} for the solution of Neuromagnetic INverse INverse problems

- Implementation
  - Basis: serial C code
  - OpenMP-tuned: blocked matrix-vector multiplication, vectorization, alignment on pages, data affinity
  - OpenMP-target: OpenMP-tuned (adapted to KNC) + target directives for offloading
  - OpenACC: up to 16 streams for parallel async. execution of kernels, pinned memory
  - CUDA: up to 16 streams, dynamic parallelism and completely asynchronous execution to minimize interaction with host, highly optimized reduction, pinned memory


NINA – TCO[7]

- **One-time costs** $C_{ot}$
  - Per node
    - HW purchase: Bull list prices from 2013 (!)
    - Building/infrastructure: as annual costs since it is amortized over 25 years
    - OS/env. installation: -
  - Per node type
    - OS/env. installation: -
    - Programming effort: Full-time employee costs 272.86 € a day

- **Annual costs** $C_{pa}$
  - Per node
    - HW maintenance: 8.2% of HW purchase costs
    - Building/infrastructure: 200,000€ per year, divided by 1.6MW, multiplied by max. power consumption of each node
    - OS/env. maintenance: 4 admins, 75% maintenance cluster (~2300 nodes): 180,000€ / 2300 = 78€ per node and year
    - Power consumption: PUE 1.5, regional electricity costs 0.15 €/kW
  - Per node type
    - OS/env. maintenance: -
    - Software/compiler: -
    - Application maintenance: - (small kernels)
NINA – Effort & Performance

**Hardware**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNB</td>
<td>2-socket Intel Sandy Bridge @ 2.7 GHz, 16 cores</td>
</tr>
<tr>
<td>K20</td>
<td>SNB + 1 NVIDIA Kepler K20x</td>
</tr>
<tr>
<td>KNC</td>
<td>SNB + 1 Intel Xeon Phi 5110p</td>
</tr>
</tbody>
</table>

productivity = outputs/inputs = \(\sum^{\text{app runs}}\) / TCO\[\$\]
**NINA – Productivity**

![Graph showing kernel runtime and effort for different programming models.](image)

**Equation:**

\[
\text{productivity} = \frac{\text{outputs}}{\text{inputs}} = \frac{\sum \text{app runs}}{\text{TCO}[$]}
\]
Suitability for Exascale
Suitability of Programming Models for Exascale

Exascale Concepts for Programming Models
Session at ISC 2016, June 19, 2016, Frankfurt

**ExaGASPI**
04:00 pm - 04:20 pm
Mirko Rahn, Fraunhofer ITWM

**MPI+X for Exascale**
04:20 pm - 04:40 pm
Bill Gropp, University of Illinois at Urbana-Champaign

**OpenMP - Taking Good Care of the Node in Exascale?**
04:40 pm - 05:00 pm
Christian Terboven, RWTH Aachen University
OpenMP vs MPI

3 : 1
Complexity of Programming Model
HPL on SGI ICE using SGI MPT

Transfer Rate only 1.63 MB/s!

Tracking down Performance Problems to individual Events
OpenMP complexity

Fork-join model

master thread → parallel region 1 → parallel region 2

fork - task a - task b - task c - task d

join - task e - task f

task g - task h - task i
OpenMP vs MPI

4 : 1
# 20 Years of OpenMP® History

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>In spring, 7 vendors and the DOE agree on the spelling of parallel loops and form the OpenMP ARB. By October, version 1.0 of the OpenMP specification for Fortran is released.</td>
</tr>
<tr>
<td>1998</td>
<td>First hybrid applications with MPI® and OpenMP appear.</td>
</tr>
<tr>
<td>1999</td>
<td>cOMPUnity, the group of OpenMP users, is formed and organizes workshops on OpenMP in North America, Europe, and Asia.</td>
</tr>
<tr>
<td>2000</td>
<td>The merge of Fortran and C/C++ specifications begins.</td>
</tr>
<tr>
<td>2001</td>
<td>Unified Fortran and C/C++: Bigger than both individual specifications combined. The first International Workshop on OpenMP is held. It becomes a major forum for users to interact with vendor.</td>
</tr>
<tr>
<td>2002</td>
<td>Minor modifications.</td>
</tr>
<tr>
<td>2003</td>
<td>Incorporates task parallelism. A hard problem as OpenMP struggles to maintain its thread-based nature, while accommodating the dynamic nature of tasking.</td>
</tr>
<tr>
<td>2004</td>
<td>Supports min/max reductions in C/C++.</td>
</tr>
<tr>
<td>2005</td>
<td>First hybrid applications with MPI® and OpenMP appear.</td>
</tr>
<tr>
<td>2006</td>
<td>The merge of Fortran and C/C++ specifications begins.</td>
</tr>
<tr>
<td>2007</td>
<td>Unified Fortran and C/C++: Bigger than both individual specifications combined. The first International Workshop on OpenMP is held. It becomes a major forum for users to interact with vendor.</td>
</tr>
<tr>
<td>2008</td>
<td>Minor modifications.</td>
</tr>
<tr>
<td>2009</td>
<td>Incorporates task parallelism. A hard problem as OpenMP struggles to maintain its thread-based nature, while accommodating the dynamic nature of tasking.</td>
</tr>
<tr>
<td>2010</td>
<td>Supports min/max reductions in C/C++.</td>
</tr>
<tr>
<td>2011</td>
<td>First hybrid applications with MPI® and OpenMP appear.</td>
</tr>
<tr>
<td>2012</td>
<td>The merge of Fortran and C/C++ specifications begins.</td>
</tr>
<tr>
<td>2013</td>
<td>Unified Fortran and C/C++: Bigger than both individual specifications combined. The first International Workshop on OpenMP is held. It becomes a major forum for users to interact with vendor.</td>
</tr>
<tr>
<td>2014</td>
<td>Minor modifications.</td>
</tr>
<tr>
<td>2015</td>
<td>Incorporates task parallelism. A hard problem as OpenMP struggles to maintain its thread-based nature, while accommodating the dynamic nature of tasking.</td>
</tr>
<tr>
<td>2016</td>
<td>Supports min/max reductions in C/C++.</td>
</tr>
<tr>
<td>2017</td>
<td>First hybrid applications with MPI® and OpenMP appear.</td>
</tr>
</tbody>
</table>

![Timeline of OpenMP developments](image_url)
Thread-local Epoch Generation (1/2)

Push Down Automaton (PDA)

- Transition depends on input symbol and stack
- PDA (stack) is required to describe the semantic of OpenMP
- In addition: output alphabet (Mealy)

9-tuple: $M = (Q, \Sigma, \Omega, \Gamma, \delta, \lambda, q_{↓0}, Z, F)$, with
- $Q = \{q_{↓m}, q_{↓p}, q_{↓h1}, q_{↓h2}, q_{↓h3}, q_{↓s}\}$ set of states
- $\Sigma$ set of OMPT events (input alphabet).
- $\Omega = \{e_{↓m}, e_{↓p}, e_{↓n}, \varepsilon\}$ output alphabet (e.g. master epoch, parallel epoch)
- $\Gamma = \{M, P, \varepsilon\}$ stack alphabet.
- $\delta = Q \times \Sigma \times \Gamma \rightarrow P(Q \times \Gamma)$ transition.
- $\lambda = Q \times \Sigma \times \Gamma \rightarrow \Omega$ output function.
- $q_{↓0} = q_{↓m}$ initial state.
- $Z = M$ initial stack symbol.
- $F = \{q_{↓m}\}$ set of accepting states.

Thread-local Epoch Generation (2/2)

- First row: Input symbol
- Second Row: Stack operation pop(a) / push(b)
- Third row(falls vorhanden): output symbol.
PDA calculation

- Example with nested parallelism:

```c
#pragma omp parallel
{
    #pragma omp parallel for
    for(...) {
        foo(...);
    }
}
```
PDA Computation (2/2) (master thread only)

- Nested parallel example code:

```c
#pragma omp parallel
{
    #pragma omp parallel for
    for(...)
    {
        foo(...);
    }
}
```

```
```

25 Analyzing Memory Accesses for Performance and Correctness of Parallel Programs

Tim Cramer
Complete PDA for OpenMP

- Additional constructs:
  - Tasking
  - Target Offloading

- More information:
OpenMP vs MPI

\[ \times \quad 3 : 1 \]
Multiparadigm programming and OpenMP

- OpenMP now supports a lot of different paradigms:
  - Threading
  - Tasking
  - Offloading
  - ...

- This is good, but complicated

- Possible rescue:
  We have to teach multiparadigm programming

---

1 Bjarn Stroustrup: “Multiparadigm programming is a fancy way of saying `programming using more than one programming style, each to its best effect.’” (Bjarn Stroustrup. FAQ)
Conclusion and outlook

• Both OpenMP and MPI are on track for Exascale
• The size and complexity of both standards are troublesome
• Multiparadigm programming is important to maintain/achieve productivity
• Programmers productivity should get more attention when developing programming models and standards

In direct comparison of the productivity of OpenMP vs. MPI:

OpenMP is the clear winner!!

.. and MPI is its best friend
Vielen Dank für Ihre Aufmerksamkeit