HPC Programming for the Future

CJ Newburn
HPC Architect, Intel

IXPUG’14 July 8-9 Austin
Caveats

- This is a forward-looking, what-if presentation
- Should not be taken as conveying our product plans
- I’m wearing my broader-community hat, not my speaking-for-Intel hat
Outline

- Some challenges
- Language interfaces
- OpenMP
- Data layout
- Library directions
Building a community

Share problems  Explore, vet, implement  Converge on standards
Some challenges

- Exposing parallelism
  - Language interfaces
  - Future proofing

- Controlling how parallelism is harvested
  - Concurrency
  - Distribution
  - Data layout

- See padalworkshop.org for forthcoming report out to broader HPC community
Layering

- **Semantic layer**
  - Describe the “what”
  - Expressiveness, intuitiveness, productivity
  - Minimalist

- **Performance control layer**
  - Describe the “how”
  - Control, efficiency
  - More pervasive
  - Achieve re-targetability through encapsulation

- **Separation of concerns**
  - Domain expert ≠ tuning expert
  - Different objectives, different rates of change, different lifetimes
Language interfaces

- **Semantic layer**
  - Less content, but it’s more stable
  - More standard, but standards change more slowly
  - Influence languages, like C++
  - Use directives that are backed by compiler support and runtime libraries, like OpenMP
  - Use functional libraries, like MKL, NumPy

- **Performance control layer**
  - More things to control → more content
  - More innovation → harder to standardize
  - Influence and develop libraries, which can change faster than compilers, like OpenMP, Kokkos
OpenMP

- **Strengths**
  - Standard, widespread, natural
  - Spans semantic and control layers

- **Weaknesses are redeemable**
  - Composability issues
  - Has some holes in its completeness

- **Transitions (see below for “→”)**
  - Improved compiler support for outer-loop parallelization
  - Offload is synchronous only → async
  - Data must be structured → decoupled from control structures
  - Constrained by C++ rules to not pass structs as parameters →
  - Each nesting layer thinks it owns the whole machine →
  - [Usage] Each library call manages OpenMP independently →
Semantic layer: expose

- Map serial specification of work onto parallel data collections
  - What to do should be separable from the order in which it’s done
  - Ex: OpenMP simd functions, lambda functions, Kokkos
  - Enrich this appropriately, e.g. with reductions, compress/expand

- Specify data reference patterns
  - Pass domain-expert knowledge to underlying system
  - Mix of reads and writes: read-only, write-once, mixed
  - Spatial locality: streaming, strided?, random
  - Temporal locality: use once, reused, persisted
  - Other: high bandwidth, working set size, etc.
Performance control layer: harvest

- **Support for parallelism**
  - SIMD/vector
  - Threads in a core
  - Cores in a node
  - Nodes in a cluster

- **Temporal**
  - Dimension order
  - Blocking
  - Work stealing

- **Binding and data layout**
  - AoS, SoA, AoSoA, ...
  - On package or not
  - Shared or distributed
Data layout challenges

- **Use of structs**
  - C++ template-based abstractions like Arrow Street, SIMD Building Blocks, maybe supported by extensions for reflection/introspection

- **Best traversal of multi-dimensional arrays**
  - Inner vs. outer loop level – directing parallelization
  - Blocking – directing traversal via insertion of nests
  - Spanning multiple access patterns – selective data re-layout
  - Abstract functions + target-tuned traversal libraries

- **Discerning access patterns**
  - Assumed-shape and pointer arrays: stride 1 or not?
  - Temporary arrays
  - MACVEC tool at TACC – LCPC submission with Jim Browne et al
  - Forthcoming Advisor/Vector Tool from Intel moving in this direction
Library directions for expansion

- **Distributed**
  - Homogeneous cluster
  - Heterogeneous cluster
  - libhta

- **Grey-box vs. black-box libraries**
  - Inlinable specialization with static guidance by users
  - Multi-phase
  - Persist distributed data
  - Decouple naming of parameters from their availability
  - OpenFOAM collaboration with Doug James
  - Parallel regions defined outside of library calls vs. within them

- **Specialized**
  - Branching in inlinable header files – error checking, special casing
  - Special sizes and shapes, with adequate motivation
“Grey-box libraries”

- **Current way**

  All inputs are in one place
  Call library, which returns all outputs
  Rinse, repeat

- **In contrast**

  Initialize: distribute, (re-)format (SpMV)
  Execute: sequence or iterate
  Mix of stable variables and updates
  Inputs and outputs may be distributed
  Overlap computes and communication
  Each partition works on its own portion
Remember...

- This is a what-if presentation, not a roadmap

- Comments on other discussions
  - 15.0 compiler has much better support for vectorization, including better support for outer loops, way better reporting
  - In 15.0, MKL headers that do error checking, specialization, and native compilation fallback in C
  - MPSS 3.3 enhanced to support MIC-MIC proxy transfers within a node, which significantly boosts bandwidth for multiple cards
Backup

- Structs as parameters
  - See my talk at padalworkshop.org
- OpenMP in a hierarchy
- Temporary array example
  - Omitted pending permission
- Challenges of abstraction
- Mapping scalar work to collections and targets
- How to specify properties
The nested OpenMP problem

- Nested but context oblivious
- Each layer thinks it owns the world (e.g. 4 wide)

```c
main() {
    omp parallel function()
    ...
}

function() {
    omp parallel for
    ...
}
```
Avoid OpenMP nesting with a hierarchy

main() {
    establish_partitions(P)
    distribute_work(P)
    sync()
}

do_work(P) {
    omp parallel for
    work(P)
}

work(P) {
    i = omp_thread_num()
    // do i^{th} part of P

- OpenMP isn’t what does the nesting
- Hierarchy established outside of OpenMP context
Challenges of Abstraction

Key challenges:

1. Defining and discovering mapping
2. Circumventing limitations posed by each domain
3. Choosing between functional and object orientation

Functional Semantics

Logical Orientation

Performant Implementation

Physical

Exposé opportunity
Convenience without enslavement
Selectively exert control
Harvest opportunity efficiently
“Scalar” work $\rightarrow \{\text{collections, targets}\}$

- Separation of concerns
- Mapping problem
- Want flexibility through abstraction, + control

<table>
<thead>
<tr>
<th>Scalar work</th>
<th>struct.x = f(...)</th>
<th>struct.y = g(...)</th>
<th>struct.z = h(...)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Data collections</th>
<th>Multi-dim arrays</th>
<th>Irregular</th>
<th>Hierarchical ...</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Ref patterns</th>
<th>Working set size</th>
<th>Access pattern</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Support for parallelism</th>
<th>SIMD/vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads in a core Nodes in a cluster ...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Temporal</th>
<th>Dimension order Blocking Work stealing ...</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Data layout</th>
<th>Array of structs (AoS) Struct of arrays (SoA) AoSoA ...</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Binding</th>
<th>On pkg or not Shared or distributed</th>
</tr>
</thead>
</table>

Chris J. Newburn, HPC Architect, Intel
PADAL Workshop, Lugano Switzerland, April 28-29 2014
# How to specify properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Data type</th>
<th>Function modifier</th>
<th>Pragma on construct</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Semantic property</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalar work</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Algorithmic freedom</td>
</tr>
<tr>
<td>Data collection</td>
<td>√</td>
<td></td>
<td></td>
<td>Kind vs. organization</td>
</tr>
<tr>
<td>Reference patterns</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>Size and logical patterns</td>
</tr>
<tr>
<td><strong>Convenience</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical data layout</td>
<td>√</td>
<td></td>
<td></td>
<td>Merge with data collection?</td>
</tr>
<tr>
<td><strong>Performance property</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical data layout</td>
<td>?</td>
<td>√</td>
<td>√</td>
<td>What’s proximate in physical arrangement</td>
</tr>
<tr>
<td>Temporal</td>
<td>?</td>
<td></td>
<td>√</td>
<td>Work order affects access patterns</td>
</tr>
<tr>
<td>Supported parallelism</td>
<td>√</td>
<td>√</td>
<td>?</td>
<td>Want data to match compute</td>
</tr>
<tr>
<td>Binding</td>
<td></td>
<td>√</td>
<td>√</td>
<td>Binding to places</td>
</tr>
</tbody>
</table>