Structured parallel programming

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Contents

- Outline of structured parallel programming
- Algorithmic skeletons
- Design patterns
- Implementation techniques
- Optimizations

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Structured parallel programming

- The concept:
  - Parallelism exploited according to (possibly nested) well known patterns
  - Rather than built from scratch on top of low level mechanisms (processes/threads, scheduling, mapping, communication, synchronization, ...)

- Complete separation of concerns
  - System programmer
    - In charge of the implementation details, including hw targeting
  - Application programmer
    - In charge of the (application/domain specific) qualitative parallelism exploitation

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Two main tracks

Algorithmic skeletons
- From HPC community
- Cole, late ‘80
- Ready to use patterns

Parallel design patterns
- From SW ENG community
- Mattson et al, early ‘00
- Implementation recipes

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
In both cases:

- **Skeletons / patterns**
  - Orchestrate parallel execution of components
  - **Components**
    - Wrappers (re-use existing sequential code)
    - Skeletons/patterns (semi-arbitrary nesting)
  - Different levels of abstraction provided to the user (the application programmer)

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Parallel design patterns

- A design pattern is a representation of a common programming problem along with a tested, efficient solution for that problem (Gamma book)

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>the problem to be solved</td>
</tr>
<tr>
<td>Context</td>
<td>the context where the pattern may be most suitably applied.</td>
</tr>
<tr>
<td>Forces</td>
<td>the different features in influencing the parallel pattern design</td>
</tr>
<tr>
<td>Solution</td>
<td>a description of one or more possible solutions to the problem solved by the pattern</td>
</tr>
</tbody>
</table>

(Mattson book)

- M. Danelutto, REPARA meeting Rapperswil, Jan 2014
# High level design spaces

## Finding concurrency space

<table>
<thead>
<tr>
<th>Decomposition</th>
<th>Task decomposition</th>
<th>Data decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency analysis</td>
<td>Group task</td>
<td>Order Task</td>
</tr>
<tr>
<td>Other</td>
<td>Design evaluation</td>
<td></td>
</tr>
</tbody>
</table>

## Algorithm structure space

<table>
<thead>
<tr>
<th>Organize by task</th>
<th>Task parallelism</th>
<th>Divide and conquer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organize by data decomp.</td>
<td>Geometric decomp.</td>
<td>Recursive data</td>
</tr>
<tr>
<td>Organize by flow of data</td>
<td>Pipeline</td>
<td>Event based coordination</td>
</tr>
</tbody>
</table>
Low level design spaces

<table>
<thead>
<tr>
<th>Supporting structures space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Program structures</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Data Structures</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation mechanisms space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UE management</strong></td>
</tr>
</tbody>
</table>
Skeletons vs. Parallel design patterns

Patterns

Skeletons

abstract pattern

implementation

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Unstructured vs structured

“From scratch” programming effort (all concerns)

Separation of concerns (functional vs. non functional)

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Algorithmic skeletons

- Parallelism exploitation patterns
  - Notable
    - Usable in a wide range of applications
  - Efficient
    - On a wide range of parallel architectures
  - Parametric
    - Functional and non-functional parameters
  - Reusable
    - Only changing parameters
  - Provided to application programmers as ready to use programming abstractions
    - Object, classes, high order functions, library entries, …

- M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Algorithmic skeletons (typical)

- **Stream parallel skeletons (computing over stream items)**

<table>
<thead>
<tr>
<th>Skeleton</th>
<th>Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task farm</td>
<td>Embarrassingly parallel, same computation over all items</td>
</tr>
<tr>
<td>Pipeline</td>
<td>Computation in stages</td>
</tr>
<tr>
<td>Feedback</td>
<td>Iterating same computation over results</td>
</tr>
</tbody>
</table>

- **Data parallel skeletons (computing over sub-items of a collection)**

<table>
<thead>
<tr>
<th>Skeleton</th>
<th>Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>Embarrassingly parallel, same computation over all sub-items</td>
</tr>
<tr>
<td>Reduce</td>
<td>Summing up collection data to scalar (associative, commutative operator)</td>
</tr>
<tr>
<td>Stencil</td>
<td>New elements of the collection from old ones + neighbours</td>
</tr>
</tbody>
</table>

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Algorithmic skeletons (typical)

- Higher level skeletons

<table>
<thead>
<tr>
<th>Skeleton</th>
<th>Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide&amp;Conquer</td>
<td>Divide problem into base cases, solve them then rebuild global solution from sub solutions</td>
</tr>
<tr>
<td>MapReduce (Google)</td>
<td>Map items to &lt;key, value&gt; and reduce values from same keys usually on big data</td>
</tr>
<tr>
<td>Branch&amp;Bound</td>
<td>Explore a tree of solutions looking for optimal ones while pruning “dead” subtrees</td>
</tr>
</tbody>
</table>
Sample algorithmic skeletons

**Ocaml**

```ocaml
Parmap.parmap: (α→β) → α list → β list
```

**SKEPU**

```plaintext
UNARYFUN(inc, float, x, return(x+1);)
skepu::Vector<float> x, y;
skepu::Map<inc> incmap(new inc);
incmap(x, y);
```

**Muesli**

```plaintext
Atomic<int,int> stage1 (compute, 1);
Pipe myPipe(stage0, stage1);
```
Sample skeleton code

- **Image filtering application**

- **Pipeline**
  - Decode video to image
  - Stencil (Filter A)
    - Local filter function
  - Farm (Filter B)
    - Filter image (g)
  - Encode image to video

---

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Skeleton composition

- Not in the original Cole work

- Simpler skeleton + composition
  - Enhance the palette of parallel patterns provided

- Two tier composition
  - Top: stream parallel skeletons
  - Middle: data parallel skeletons
  - Bottom: sequential wrappers

---

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Implementation

- Template based (FastFlow, SKEPU, Muesli, SkeTo)
  - Multiple visit of the skeleton tree
  - Assignment of existing templates, properly instantiated, to each skeleton
  - Optimising non functional properties (e.g. parallelism degree, mapping, …)

- Macro data flow based (Muskel, Skipper, Skandium)
  - Skeleton tree to MDF graph
  - MDF graph to parallel MDF interpreter

- Static vs. dynamic optimization tradeoff

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Optimizations

- **Rewriting rules**
  - Farm(S) = S
  - Pipe(A, B) = Comp(A, B)
  - Pipe(Map(A), Map(B)) = Map(Pipe(A), Pipe(B))
Optimizations

- Automatically devising non functional parameters

- E.g. Pipeline(Farm(A), Farm(B))
  - Ta and Tb known
  - Pardegree(Farm(A)) : ParDegree(Farm(B)) = Ta : Tb
  - Bounded by
    - Available resources
    - Inter arrival time
Offloading

- Known parallel task structure
  - Decide (schedule) tasks to offload to accelerators
  - Automatically manage data transfers and computation scheduling
  - Profit of different kinds of accelerators in different skeletons

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Optimizations

- Optimal allocation of data parallel tasks to CPU and GPU cores
  - Depending on
    - PCIe bandwidth
    - I/O data sizes
    - Kernel execution time
  - Joint exploitation of both CPU and GPU resources
    - Optimizing completion time
    - And, possibly, power consumption

- Similar results for cloud offloading of excess parallel computations

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Skeleton workflow

- Program coding
  - Composition of abstract skeletons from a palette

- Compiling
  - Assignment of implementations (from library) to skeletons
  - Devising proper non functional parameters (e.g. par degree)

- Optimization
  - Target driven rewriting

- Code generation
  - High level code + library calls

Application programmer concern

System programmer concern

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
A little bit of history

1990
• Algorithmic skeletons introduced
• Skeleton languages (P3L, Fortran M) and libraries (Ocam!P3L, Muesli)
• Mostly from HPC community, poor success

2000
• OO skeleton programming framework (Muesli, Muskel, SkeTo, ASSIST)
• Design patterns introduced – Parallel design patterns introduced
• More libraries available (C/C++, ML, Java, …)
• Google mapreduce

2010
• Mature skeleton technology (FastFlow, SKEPU, Muesli)
• Parallel design patterns recognized in multiple contexts (TBB, Microsoft TPL)
• Google (Pregel, BSP, mapreduce optimization (Flume), …)
• Co-processors targeted (SKEPU, SkeCL, Muesli, FastFlow)

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Perspectives

- Google
  - Map reduce, composition optimizations, BSP on large data
- Intel TBB
  - Common and low level patterns as C++ abstractions
- Microsoft TPL
  - Stream and data parallel patterns in C#
  - Including data flow computing
- HPC community
  - More and more projects
- Sw Engineering community
  - Building new programming models (+ formal tools)

M. Danelutto, REPARA meeting Rapperswil, Jan 2014
Summary/key concepts

- Parallelism = skeletons (and only skeletons)
- Implementation guarantees
  - Hardware targeting & efficiency
  - Correctness
  - Other non-functional properties (e.g. fault tolerance, power management, etc.)
- Complete separation of concerns
  - Application programmer => qualitative parallelism exploitation
  - System programmer (once and for all) => quantitative & hw dependent aspects, including portability (functional and non-functional)

M. Danelutto, REPARA meeting Rapperswil, Jan 2014