Second generation skeleton systems

Marco Danelutto
Dept. Computer Science
University of Pisa
Contents

• Skeletons
• The Pisa experience
• Cole’s manifesto
• Additions to Cole’s manifesto
• muskel
• ASSIST
• Conclusions
Focus of the talk

• Assume you know skeletons ...
• Focus on skeleton concept evolution
  – Concept
  – Implementation
  – Results
• Report on Pisa experience
  – Working on skeletons since 1990
  – Several prototypes / environments
  – Several lessons learned (!)
Skeletons

• Known, efficient, common parallelism exploitation patterns
  – AKA: templates, design patterns, coordination patterns, components, ...
• Since late ‘80 (Cole’s PhD thesis)
• Range of flavors
  – Languages/libraries
  – Functional/imperative
  – Composable/flat
Evolution

Cole PhD (1988)
Fixed degree DC, Iterative combination, Cluster Task queue

Darlington (1992)
Pipeline, Farm, RaMP, DMPA

P3L (1991)
Pipeline, Farm, Map, Reduce

BMF ('80)
map fold reduce prefix + algebra

La Laguna ('90)
Advance perf modeling

Kuchen Skil (1998)

Serot (1999)
Skipper (→MDF)

Kuchen Muesli (2002)
Pipeline, Farm, Parallel array + collectives

Skellicorn (mid '90)

Gorlatch (late '90)

SkeTo (early '00)

HOC (early '00)

eSkel (2002)
Parametric skeletons + Give/Take

PARCO'05 MiniSymposium Algoritmic Skeletons and High-level Concepts for Parallel Programming
Malaga, Sept. 13th, 2005 “Second generation skeleton system” M. Danelutto(marcod@di.unipi.it)
The Pisa picture

P3L (the Pisa Parallel Programming Language 1991)

SkIE
(Skeleton Integrated Environment 1997)

OcamlP3L
(1998)

Macro Data Flow RunTime (1999)

SKElab (SKEleton LIBrary 2000)

Lithium (2000)

ASSIST
(A Software development System based on Integrated Skeleton Technology 2001)

muskel
(μskeleton lib 2003)
The Pisa picture: influences ...

P3L (the Pisa Parallel Programming Language 1991)

SkIE (Skeleton Integrated Environment 1997)

OcamlP3L (1998)

Muesli

Macro Data Flow RunTime (1999)

Lithium (2000)

HOC

muskel (μskeleton lib 2003)

eSkel

ASSIST (A Software development System based on Integrated Skeleton Technology 2001)
The Pisa picture: alive projects

P3L (the Pisa Parallel Programming Language 1991)

SkIE
(Skeleton Integrated Environment 1997)

OcamlP3L
(1998)

Macro Data Flow RunTime (1999)

SKElib (SKEleton LIBrary 2000)

Lithium (2000)

ASSIST
( A Software development System based on Integrated Skeleton Technology 2001)

muskel
(μskeleton lib 2003)
The Pisa picture: side projects

ASSIST
(A Software development System based on Integrated Skeleton Technology 2001)

Macro Data Flow RunTime (1999)

Lithium (2000)

muskel (µskeleton lib 2003)

JXTAskel

dynamic ASSIST

ProActive Skel Components

JJPF
Library vs. Language

P3L (the Pisa Parallel Programming Language 1991)

SkIE (Skeleton Integrated Environment 1997)

ASSIST (A Software development System based on Integrated Skeleton Technology 2001)

Macro Data Flow RunTime (1999)

OcamlP3L (1998)

SKElib (SKEleton LIBrary 2000)

Lithium (2000)

muskel (μskeleton lib 2003)

PARCO’05 MiniSymposium Algorithmic Skeletons and High-level Concepts for Parallel Programming
Malaga, Sept. 13th, 2005 “Second generation skeleton system” M. Danelutto(marcod@di.unipi.it)
Template vs. Macro Data Flow

P3L (the Pisa Parallel Programming Language 1991)

SkIE
(Skeleton Integrated Environment 1997)

OcamlP3L
(1998)

Macro Data Flow RunTime (1999)

SKElib (SKEleton LIBrary 2000)

Lithium (2000)

ASSIST
(A Software development System based on Integrated Skeleton Technology 2001)

muskel
(μskeleton lib 2003)
Fixed Skeleton Set vs. Parametric

P3L (the Pisa Parallel Programming Language 1991)

SkIE
(Skeleton Integrated Environment 1997)

OcamlP3L
(1998)

Macro Data Flow RunTime (1999)

SKElib (SKEleton LIBrary 2000)

Lithium (2000)

ASSIST
(A Software development System based on Integrated Skeleton Technology 2001)

muskel
(μskeleton lib 2003)
Code reuse vs. from scratch

P3L (the Pisa Parallel Programming Language 1991)

SkIE (Skeleton Integrated Environment 1997)

OcamlP3L (1998)

Macro Data Flow RunTime (1999)

SKElib (SKEleton LIBrary 2000)

Lithium (2000)

ASSIST (A Software development System based on Integrated Skeleton Technology 2001)

muskel (μskeleton lib 2003)

PARCO’05 MiniSymposium Algorithmic Skeletons and High-level Concepts for Parallel Programming
Malaga, Sept. 13th, 2005  “Second generation skeleton system” M. Danelutto(marcod@di.unipi.it)
Interoperability

P3L (the Pisa Parallel Programming Language 1991)

SkIE (Skeleton Integrated Environment 1997)

OcamlP3L (1998)

Macro Data Flow RunTime (1999)

SKElib (SKEleton LIBrary 2000)

Lithium (2000)

ASSIST (A Software development System based on Integrated Skeleton Technology 2001)

muskel (μskeleton lib 2003)

PARCO'05 MiniSymposium Algoritmic Skeletons and High-level Concepts for Parallel Programming
Malaga, Sept. 13th, 2005  “Second generation skeleton system” M. Danelutto(marcod@di.unipi.it)
Heterogeneity

PARCO'05 MiniSymposium Algorithmic Skeletons and High-level Concepts for Parallel Programming
Malaga, Sept. 13th, 2005

"Second generation skeleton system"  M. Danelutto(marcod@di.unipi.it)
Dynamcity

P3L (the Pisa Parallel Programming Language 1991)

SkIE
(Skeleton Integrated Environment 1997)

OcamlP3L
(1998)

Macro Data Flow RunTime (1999)

SKElib (SKEleton LIBrary 2000)

Lithium (2000)

ASSIST
(A Software development System based on Integrated Skeleton Technology 2001)

muskel
(µskeleton lib 2003)
Lessons learned (before manifesto)

• Tools that allow code reuse have better agreement
• Tools that interact with “standard” environments have better agreement
• Programmers always want just a little bit more parallelism exploitation forms
• Libraries are easier to sell than languages
• Modern architectures must be targeted
  – Heterogeneous, dynamic, non exclusive usage, etc.
Cole’s manifesto

1. Propagate the concept with minimal disruption
   - No chance to introduce yet another parallel programming language

2. Integrate ad hoc parallelism
   - Specialized, ad hoc solutions must be hosted

3. Accommodate diversity
   - Slightly different skeletons should be derivable

4. Show the payback
   - Advertising: demonstrate that moving to skeletons is worthwhile
Our additions

5 Support code reuse
   - Huge amount of (dusty deck?) code
   - Large amounts of (open source) “libraries”

6 Handle heterogeneity
   - Cluster/networks/grids are heterogeneous
   - Upgrades of clusters (with different release procs and different amounts/speed of main store)

7 Handle dynamicity
   - Non dedicated computing nodes (varying load)
   - Different nodes, different power
Our claim

Skeleton systems (either skeleton based languages or skeleton libraries) can be classified as “mature, second-generation” if they satisfy the 7 requirements:

1. Minimal disruption
2. Ad hoc parallelism
3. Accommodate diversity
4. Show payback
5. Reuse code
6. Handle heterogeneity
7. Handle dynamicity
Current skeletons in Pisa

- Learn from past experiences
- Address both Cole’s manifesto and our additional requirements
- One experimental framework (muskel)
  - Compact, easy to modify, research only
  - Exploits Java to shortcut some problems (portability)
- One production framework (ASSIST)
  - Product of the GRID.it project
  - Huge, interoperable, runs on standard environments
muskel

- μ-skeleton library → muskel
- Full Java library (1 5)
- Small(est) subset of Lithium
  - Macro data flow (MDF) (2 3)
  - RMI based remote execution
- P2P-like resource recruiting
  - UDP multicast (→ ?JXTA, → ?ProActive)
- With application manager
  - Ensuring user provided performance contract
  - Reacts to network / node faults
  - Looking for new resources
  - Rescheduling lost MDF instructions from scratch
Parameter sweeping application
Parameter sets on disk files
Set of remote WS accessible (any OS with Java)

Target: get a set of result files in the shorter time

1. Minimal disruption
2. Ad hoc parallelism
3. Accommodate diversity
4. Show payback
5. Reuse code
6. Handle heterogeneity
7. Handle dynamicity
• On remote machines, run muskel run time (once and forall)
  – RemoteWorker java RMI server

• On the local machine just run (1 5):

```java
public static void main (String [] args)
{
    Compute mainProgram = new Farm(new doSweep());
    ParDegree parDegree =
        new ParDegree(Integer.parseInt(args[0]));
    ApplicationManager manager = new ApplicationManager(mainProgram);
    manager.setContract(parDegree);
    manager.inputStream(args[1]);
    manager.evalToFile(args[2]);
}
```

1. Minimal disruption
2. Ad hoc parallelism
3. Accommodate diversity
4. Show payback
5. Reuse code
6. Handle heterogeneity
7. Handle dynamicity

“Second generation skeleton system” M. Danelutto (marcod@di.unipi.it)
Compute mainProgram = new Farm(new doSweep()); \hspace{1cm} \textit{define program}

ParDegree parDegree =
new ParDegree(Integer.parseInt(args[2])); \hspace{1cm} \textit{prepare the perf contract to ask}

ApplicationManager manager = new ApplicationManager(mainProgram); \hspace{1cm} \textit{instantiate a manger and tell it the program to be computed}

manager.setContract(parDegree); \hspace{1cm} \textit{provide required performance contract}

manager.inputStream(args[0]); \hspace{1cm} \textit{tell the manager where to find input tasks}

manager.evalToFile(args[1]); \hspace{1cm} \textit{ask for program evaluation (res to file)}
**muskel implementation**

- **Basics**
  - Centralized task pool hosts MDF instructions
  - One thread per remote worker:
    - Fetches a fireable instruction, invokes remote method to compute it, delivers resulting tokens to the proper place
  - Operates in **normal form** mode
    - Skeleton tree preprocessed to normal form, service time optimized

- **Remote workers**
  - Methods to accept MDF code (serialized)
  - Methods to compute MDF
  - Management methods (stats, load measure, etc.)
muskel implementation (2)

- Application manager (7!!!)
  - Accepts performance contract
    - Parallelism degree (current)
    - Service time (forthcoming)
  - Discovers available resources
    - UDP multicast (current)
    - P2P (forthcoming)
  - Ensures fault tolerance
    - Faulty nodes replaced & tasks rescheduled
  - Ensures QoS
    - Nodes added if needed
    - Nodes released if steady over-contract

- Minimal disruption
- Ad hoc parallelism
- Accommodate diversity
- Show payback
- Reuse code
- Handle heterogeneity
- Handle dynamicity
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propagate the concept with minimal disruption</td>
<td>Plain Java library</td>
</tr>
<tr>
<td>Integrate <em>ad hoc</em> parallelism</td>
<td>User access to streams &amp; MDF level</td>
</tr>
<tr>
<td>Accommodate diversity</td>
<td>User access to streams &amp; MDF level</td>
</tr>
<tr>
<td>Show the pay back</td>
<td>OO + rapid prototyping + efficiency</td>
</tr>
<tr>
<td>Support code reuse</td>
<td>Java only</td>
</tr>
<tr>
<td>Handle heterogeneity</td>
<td>By Java</td>
</tr>
<tr>
<td>Handle dynamicity</td>
<td>Application manager</td>
</tr>
</tbody>
</table>

**Muskel & requirements**

PARCO’05 MiniSymposium *Algoritmic Skeletons and High-level Concepts for Parallel Programming*

Malaga, Sept. 13th, 2005  “Second generation skeleton system”  M. Danelutto(marcod@di.unipi.it)
ASSIST

• Generic graph of either parallel or sequential modules
  – Sequential modules: C, C++, F77, (Java) (⑥)
  – Parallel modules (② ③):
    • Set of virtual processes (VP, named after a topology)
    • Program of the virtual processes as a seq module
    • (Possibly) sharing a state
    • Processing state and input data
    • To produce output data
    • SPMD is a sub case

• Interconnected via data flow streams
  – Non deterministic control over input streams
  – Input data scatter/uni-multi-broadcast to VPs

① Minimal disruption
② Ad hoc parallelism
③ Accommodate diversity
④ Show payback
⑤ Reuse code
⑥ Handle heterogeneity
⑦ Handle dynamicity
ASSIST (2)

- Interoperability (5)
  - With WS & CORBA/CCM
  - To & From
    - ASSIST programs wrapper to CCM/WS (compile flag!)
    - CCM/WS accessible from within ASSIST programs
      (as modules in the graph or called from seq and VPs)
- GRID.it component model on top of ASSIST
  - Pipeline, farm, generic graph component
- Module (parmod) + application managers (7)
- Runs on top of (6)
  - Cluster/networks of TCP/IP POSIX workstations
  - Globus 2.4 grids
ASSIST sample code (1)

generic main() {
    stream Param_t params;
    stream Res_t res;
    generate_instream (output_stream params);
    doSweep(input_stream params output_stream res);
    process_outstream (input_stream res);
}

#pragma pardegree doSweep N

parmod doSweep(input_stream Param_t param output_stream Res_t result) {
    topology none my_vp;
    do input_section {
        guard1: on true, MAX_PRI, param { distribution param on_demand to my_vp; }
    } while (true)
}

virtual_processes {
    computeSweep(in guard1 out result) {
        VP { doSweepSeq(in param out result); }
    }
}

output_section { collects result from ANY my_vp; }

Define parallel module graph
Set perf contract
Name processes (none = task farm)
Schedule input tasks
Define parallel module
Define logically parallel activities
Deliver computed results
ASSIST sample code (2)

```c
proc generate_instream (output_stream Param_t params)
$c{  Param_t p;
    ...
    p = ...;
    assist_out(params,p);  }
c$

proc process_outstream(input_stream Res_t res)
$inc <iostream>
$c{  //  ... some code processing res here ...
  }
c$

proc doSweepSeq(in Param_t p out Res_t r)
$objc<myLibF-1.2.so>
$c{  r = f(p);
  }
c$
```

Specify seq language
Deliver items to output stream
Includes ...
Link objects and libs ...

PARCO'05 MiniSymposium *Algoritmic Skeletons and High-level Concepts for Parallel Programming*
Malaga, Sept. 13th, 2005    “Second generation skeleton system”  M. Danelutto(marcod@di.unipi.it)
ASSIST implementation
### ASSIST & requirements

<table>
<thead>
<tr>
<th>Number</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Propagate the concept with minimal disruption</td>
<td>... definitely NO!</td>
</tr>
<tr>
<td>2</td>
<td>Integrate <em>ad hoc</em> parallelism</td>
<td>Parametric parmod</td>
</tr>
<tr>
<td>3</td>
<td>Accommodate diversity</td>
<td>Parametric parmod</td>
</tr>
<tr>
<td>4</td>
<td>Show the pay back</td>
<td>Fairly fast application development + high efficiency</td>
</tr>
<tr>
<td>5</td>
<td>Support code reuse</td>
<td>C, C++, Fortran 77</td>
</tr>
<tr>
<td>6</td>
<td>Handle heterogeneity</td>
<td>Compiler + run time</td>
</tr>
<tr>
<td>7</td>
<td>Handle dynamicity</td>
<td>Module &amp; application manager</td>
</tr>
</tbody>
</table>

"Second generation skeleton system"  
*M. Danelutto (marcod@di.unipi.it)"
## A final comparison

<table>
<thead>
<tr>
<th></th>
<th>muskel</th>
<th>ASSIST</th>
<th>eSkel</th>
<th>musesi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Propagate the</td>
<td>Plain Java library</td>
<td>...</td>
<td>Plain MPI</td>
<td>Plain C++ &amp; MPI library</td>
</tr>
<tr>
<td>concept with</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>minimal disruption</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Integrate ad hoc</td>
<td>User access to streams &amp; MDF</td>
<td>Parametric parmod</td>
<td>Protected MPI</td>
<td>Variety of combinations of</td>
</tr>
<tr>
<td>parallelism</td>
<td>level</td>
<td></td>
<td>communicators within skeletons</td>
<td>(data parallel) skeletons</td>
</tr>
<tr>
<td>3 Accommodate diversity</td>
<td>User access to streams &amp; MDF</td>
<td>Parametric parmod</td>
<td>Parametric skeleton calls</td>
<td></td>
</tr>
<tr>
<td></td>
<td>level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Show the pay back</td>
<td>OO + rapid prototyping +</td>
<td>Fairly fast application</td>
<td>Fast application development</td>
<td>OO library expressive power +</td>
</tr>
<tr>
<td></td>
<td>efficiency</td>
<td>development + high efficiency</td>
<td></td>
<td>fast development</td>
</tr>
<tr>
<td>5 Support code</td>
<td>Java only</td>
<td>C C++ Fortran77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Handle heterogeneity</td>
<td>By Java</td>
<td>Compiler + run time</td>
<td>Guaranteed by MPI</td>
<td>Guaranteed by MPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 Handle dynamicity</td>
<td>Application manager</td>
<td>Module &amp; application manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PARCO'05 MiniSymposium *Algorithmic Skeletons and High-level Concepts for Parallel Programming*
Malaga, Sept. 13th, 2005    “Second generation skeleton system” M. Danelutto(marcod@di.unipi.it)
Conclusions

• **Summary**
  – Large experience with skeletons

• **Requirements**
  – Extending Cole’s ones

• **Current experiences in Pisa**
  – Outlined
  – Related to requirements
  – And with other acknowledged skeleton systems
any questions?

marcod@di.unipi.it

www.di.unipi.it/~marcod
(links to both muskel and ASSIST home pages)
(copy of these slides (soon))