Macro Data Flow execution model

Marco Danelutto
Dept. Computer Science
University of Pisa
Italy

CoreGRID
Programming model Institute
Summary

• Macro Data flow

• Skeletons to macro data flow

• Workflows

• Distributed macro data flow interpreter

• MDF for grids

• Conclusions
Macro Data flow

- Data flow
  - graphs of data flow instructions
  - tokens
  - fireable instructions
  - handling state
  - handling stream computations
Macro Data flow

- Data flow
  - graphs of data flow instructions
  - tokens
  - fireable instructions
- handling state
  - handling stream computations
Macro Data flow

• Data flow
  • graphs of data flow instructions
  • tokens
  • fireable instructions
  • handling state
  • handling stream computations
Macro Data flow

- Data flow
  - graphs of data flow instructions
  - tokens
  - fireable instructions
  - handling state
  - handling stream computations
Macro Data flow

- Data flow
  - graphs of data flow instructions
  - tokens
  - fireable instructions
  - handling state
  - handling stream computations
Macro Data flow

- Data flow
  - graphs of data flow instructions
  - tokens
  - fireable instructions
  - handling state
  - handling stream computations
Macro Data flow

- Data flow
  - graphs of data flow instructions
  - tokens
  - fireable instructions
  - handling state
  - handling stream computations
Macro Data flow

- Data flow
  - graphs of data flow instructions
  - tokens
  - fireable instructions
  - handling state
  - handling stream computations
Macro Data Flow

- Code computed in instructions
  - sensibly bigger that a single instruction, possibly entire chucks of code
- Conceivable as plain sequential code
  - clearly identified input and output parameters
- any kind of code, provided there is some common primitive data representation
Translating to MDF : skeletons

- Danelutto 1999, PARCO paper
  - Skipper (J. Serót 2002)
  - Lithium (Teti, Danelutto, Aldinucci 2002) and muskel (Danelutto, 2004)

- compile(seq(S)) => mdfi(S)
- compile(pipe(A,B)) => Ga = compile(A), Gb = compile(B), concat(Ga,Gb)
- compile(farm(A)) => compile(A)

∀ input task Ti instantiate G(Ti) and run it on the (distributed) interpreter
Translating to MDF: workflows

• Naive translation
  • each node in the graph corresponds to a single MDFi

• Works under two assumptions
  • serializability of the code
  • stream parallel computations (if #stream=1 then limited advantages)
Distributed MDF interpreter

- Centralized instruction pool
  - thread instantiating graphs with input tokens (tasks)
  - n threads feeding remote PEs (1 per PE)
    - fetch fireable, dispatch, get results, route results, loop
  - code to be executed in the MDF is serialized first, once and for all
- Remote interpreter instance
  - provides a `Object compute(Object)` method + stats and discovery
Distributed MDF interpreter

MDFi repository
Distributed MDF interpreter

thread: feed input tokens → MDFi repository
Distributed MDF interpreter

thread: feed input tokens

MDFi repository

thread: fetch fireable, RMI
compute, route result

thread: fetch fireable, RMI
compute, route result

thread: fetch fireable, RMI
compute, route result

thread: fetch fireable, RMI
compute, route result
Distributed MDF interpreter

thread: feed input tokens

MDFi repository

Remote PE
Remote PE
Remote PE
Remote PE

thread: fetch fireable, RMI compute, route result
thread: fetch fireable, RMI compute, route result
thread: fetch fireable, RMI compute, route result
thread: fetch fireable, RMI compute, route result
Distributed^2 MDF interpreter

• Decentralized MDFi repository

  • tree, ring

• processes in the distributed MDFi repository nodes

  • distribute excess instructions to other instances if over a high water mark and ask for fireable instructions if under a low water mark

  • other strategies of work stealing can be implemented

• remote interpreters directly associated to MDFi repositories

  • hopefully most communication are local
Distributed² MDF interpreter
Distributed² MDF interpreter
Distributed\textsuperscript{2} MDF interpreter

thread: feed input tokens

Pool\textsubscript{i}

Excess Fireable MDF\textsubscript{i}

Control CMD (requests)

Pool\textsubscript{i}

MDF interpreter instance

MDF interpreter instance

MDF interpreter instance
Distributed\(^2\) MDF interpreter (2)
Distributed² MDF interpreter (2)
Distributed² MDF interpreter (2)

- thread: feed input tokens
  
  Poolᵢ → Poolᵢ → Poolᵢ
  
  Excess Fireable MDFᵢ
  
  Control CMD (requests)
Distributed² MDF interpreter (2)

thread: feed input tokens

Poolᵢ

Excess Fireable MDFᵢ

Control CMD (requests)

Poolᵢ

Poolᵢ

Poolᵢ
Distributed\(^2\) MDF interpreter (2)
Distributed² MDF interpreter (2)

thread: feed input tokens

"Pool" i

Excess Fireable MDFi

Control CMD (requests)

Pool i

Pool i

Pool i

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
Distributed\(^2\) MDF interpreter (2)

thread: feed input tokens

Pool\(_i\)

Excess Fireable MDF\(_i\)

Control CMD (requests)

Pool\(_i\)

Pool\(_i\)
Distributed$^2$ MDF interpreter (2)
Distributed$^2$ MDF interpreter (2)

thread: feed input tokens

Pool

Excess Fireable MDF\textsubscript{i}

Control CMD (requests)

Pool\textsubscript{i}

Pool\textsubscript{i}

Pool\textsubscript{i}

High water mark

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
Distributed$^2$ MDF interpreter (2)

- Thread: feed input tokens
- Pool$_i$
- Excess Fireable MDF$_i$
- Control CMD (requests)
- Pool$_i$
- Pool$_i$
- Pool$_i$
- High water mark
Distributed$^2$ MDF interpreter (2)
Distributed² MDF interpreter (2)
Current prototype **muskel**

- Java library
  - RMI based remote interpreters
  - UDP multicast for discovery
  - MDFi storage serialized at the very beginning
- Classical stream parallel skeletons: farm & pipelines
- **Compute** wrapper for sequential code & **ParCompute** special wrapper for arbitrary macro data flow code
  - possibility to implement any kind of skeleton
muskel
muskel
public class Map2 extends ParCompute {
    public Map2(Compute f, Manager manager) {
        program = new MdfGraph();
        Dest [] dds1 = new Dest[2];
        dds1[0]=new Dest(0,2);
        dds1[1]=new Dest(0,3);
        Mdfi emitter = new Mdfi(manager,1,new MapEmitter(2),1,2,dds1);
        program.addInstruction(emitter);
        Dest [] dds2 = new Dest[1];
        dds2[0] = new Dest(0,4);
        Mdfi if1 = new Mdfi(manager,2, f, 1, 1, dds2);
        program.addInstruction(if1);
        Dest [] dds3 = new Dest[1];
        dds3[0] = new Dest(1,4);
        Mdfi if2 = new Mdfi(manager,3, f, 1, 1, dds3);
        program.addInstruction(if2);
        Dest[] ddslast = new Dest[1];
        ddslast[0] = new Dest(0,Mdfi.NoInstrId);
        Mdfi coll = new Mdfi(manager,4,new MapCollector(),2,1,ddslast);
        program.addInstruction(coll);
        return;
    }
}
```java
public static void main(String[] args) {
    Manager manager = new Manager();

    Compute seqStage = new IncDoubleVector();
    Compute worker    = new Fdouble();
    Compute mapStage  = new Map2(worker, manager);
    Pipeline main     = new Pipeline(mapStage, seqStage);

    InputManager inManager = new DoubleVectIM(5, 4); // 5 tasks (#=4)
    OutputManager outManager = new DoubleVectOM();
    ParDegree contract = new ParDegree(Integer.parseInt(args[0]));
    manager.setInputManager(inManager);
    manager.setOutputManager(outManager);
    manager.setContract(contract);
    manager.setProgram(main);
    manager.compute();
}
```
muskel
for large scale distributed, GRID and Peer-to-Peer Technologies
**muskel on grids**

- Current port of **muskel** on top of ProActive
  
  - without firewalls: remote interpreters migrateTo remote nodes discovered through multicast or XML config files
  
  - with firewalls: remote interpreters launched via descriptors exploiting ssh tunneling

- Interpreter unchanged

- advantages retained

- small additional overhead, mainly for launching and ssh tunneling
muskel on grids (security)

**muskel** on grids (security 2)
Task 3.3 connection

• Worth to investigate new models on top of the component model

  • remote interpreter component + custom front end parser/compiler components

• Worth to compare with other skeleton approaches

  • HOC ?

• Suitable to host several different high level programming models on top of

  • Open problems: affinity scheduling, token caching, $D^2$ implementation
Conclusions

• Fairly simple execution model

• Works with serializable code

• Suitable to implement several higher level models

• Call for cooperation !
Credits

• MDF model
  • UNIPI (M. Danelutto, M. Aldinucci)

• expandability
  • UNIPI (M. Danelutto) + ISTI/CNR (P. Dazzi)

• Security
  • UNIPI (M. Aldinucci)