‘Daedalus’
Automated parallelization of sequential programs and programming of parallel systems

dr. Hristo Nikolov
LIACS
Leiden University

Leiden, Fall 2011
Parallel computing systems are everywhere

Multi-core/multi-processor systems

General purpose

Mobile and embedded
Muti-processor/-core systems
Network-on-Chip

Intel’s 48-core chip
Why parallel systems?

- **Single Core A9**
  - Single Core CPU 100% utilized due to workload
  - Needs to run at:
    - Max Frequency = 1 GHz
    - Max Voltage = 1.1V
  - Consumes Power = P

- **Dual Core A9**
  - Two cores share workload
  - Each core 50% utilized
  - Cores run at lower frequency:
    - Frequency = 550 MHz
    - Voltage = 0.8V
  - Consumes Power = 0.6P

**40% Lower Power than Single Core CPU**
How to Design and Program Parallel systems?
Programming Problem

EASY to specify
Sequential Application Specification

```c
for(j=1;j<=N;j++) {
    x[j] = Source1();
}
for(i=1;i<=K;i++) {
    y[i] = Source2();
}
for(j=1;j<=N;j++) {
    for(i=1;i<=K;i++) {
        y[j]=F(y(i),x(j));
    }
}
for(i=1;i<=K;i++) {
    out[i]=Sink(y(i));
}
```

DIFFICULT to map

DIFFICULT to specify
Parallel Application Specification

EASY to map

Compiler
The Daedalus Design-flow

High-level Models

Library IP Cores

RTL-level Models

System-level DSE (Sesame)

Platform specification

Mapping specification

Polyhedral (Kahn) Process Network

System-level synthesis (Espam)

Synthesizable VHDL

MP-SoC

C/C++ code for processors

Commercial Compilers/HW Design tools

C program

pnGen

RTL Level

Synthesizable VHDL

MP-SoC

C/C++ code for processors

Commercial Compilers/HW Design tools
Programming Problem

EASY to specify
Static Affine
Nested Loop Programs

```
for(j=1;j<=N;j++) {
    x[j] = Source1();
}
for(i=1;i<=K;i++) {
    y[i] = Source2();
}
for(j=1;j<=N;j++) {
    for(i=1;i<=K;i++) {
        y[j]=F(y(i),x(j));
    }
}
for(i=1;i<=K;i++) {
    out[i]=Sink(y(i));
}
```

DIFFICULT to map

DIFFICULT to specify
Polyhedral
Process Networks

Application

pnGen tool

EASY to map

ESPAM tool
Automated Parallelization of sequential Static Affine Nested-loop Programs to Polyhedral Process Networks
Static Affine Nested Loop Programs

- Restrictions to input top-level program
  - Parameters are *symbolic constants*, i.e., do not change at run-time
  - *Only FOR-loops* with *bounds that are affine functions* of other loops’ indices and parameters
  - *Only If-statements* with *conditions that are affine functions* of loops’ indices and parameters
  - *Only Explicit data exchange* via arrays or scalars (*NO pointers*). Arrays are *indexed with affine functions* of loops’ indices and parameters
  - *NO restrictions* for code in function calls

```c
int N = 10;
#pragma parameter N 5 100

void main(void) {
    int i, j;
    int A[600];

    for (j=1; j<=N; j++) {
        for (i=j; i<=3*j-2; i++) {
            if ( i+j<=4*N-6 ) {
                A[i] = F1();
            }
        }
    }
}
```
Polyhedral Process Networks (1)

- Simple formalism to express concurrency
  - Autonomously running processes
  - Communicating via bounded FIFOs
  - Synchronization via blocking read/write

- Deterministic
  - For one and the same input, one and the same output is produced, irrespective of the execution order of processes

- Distributed Control
  - No global schedule needed
Polyhedral Process Networks (2)

- Well defined structure of a process
  - READ - EXECUTE – WRITE sections
  - CONTROL – determines the number of iterations/firings of a process
- Every Process, Input and Output Port can be represented as Parameterized Polyhedrons
- Parameterized Polyhedron
  \[ \mathcal{P}(p) = \{ x \in \mathbb{Q}^n \mid Ax = Bp + b \land Cx \geq Dp + d \} \]
  - Set of points \( x \) in the \( n \)-dimensional space satisfying some affine constraints
  - \( p \in \mathbb{Q}^m \) is a vector of parameters

```c
// Process F2
void main(){
    for(j=2; j<=N; j++){
        for(i=j; i<=3*j-2; i++){
            if(j-2 == 0)
                read(IP1,in_0);
            if(j-3 >= 0)
                read(IP2,in_0);
            F2(in_0,out_0);
            if(-j+N-1 >= 0)
                write(OP1,out_0);
            if(j-N == 0)
                write(OP2,out_0);
        }
    }
}
```
Polyhedral Process Networks (3)

Example: Input port IP2 as parameterized polyhedron

\[ P(M, N) = \left\{ (i, j) \in \mathbb{Z}^2 \mid \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \geq \begin{bmatrix} 0 & 0 \\ -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} M \\ N \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \\ 3 \\ 0 \end{bmatrix} \right\} \]

// Process
void main(){
for(i=2; i<=M; i++){
  for(j=2; j<=N; j++){
    if(j-2 == 0)
      read(IP1, in_0);
    if(j-3 >= 0)
      read(IP2, in_0);
  }
}

// Read
if(j-2 == 0)
  read(IP1, in_0);
if(j-3 >= 0)
  read(IP2, in_0);
Deriving PPNs

Sequential Program

int N = 5;
#pragma parameter N 4 16;
int K = 100;
#pragma parameter K 10 100;

for (k=1; k<=K; k++)
    for (j=1; j<=N; j++) {
        t = C(r[j][j], x[k][j], r[j][j], x[k][j]);
        for (i = j+1; i<=N; i++) {
            t = D(t, r[j][i], x[k][i], r[j][i], x[k][i]);
        }
    }

Dependence Analysis

PDG

Polyhedral Dependence Graph

Linearization

Polyhedral Process Network

FIFO size calculation

Polyhedral Process Network (optimized)
Dataflow Analysis (1)

Static affine program

```
for( j=1; j<=4; j++ ) {
    x[j] = F1( ... );
}
for( j=1; j<=4; j++ ) {
    if( j<=2 ) {
        x[j] = F2( x[j] );
    }
    ... = F3( x[j] );
}
```

Exact Array Dataflow Analysis

**Q:** Given a read from an array element, what was the last write to that array element?  
   **Example:** given a read $<F3 @ j=2>$ from $x(2)$ what was the last write $<F? @ j=?>$ to $x(2)$  

**A:** Can be computed using **Parametric Integer Programming (PIP)**  
   $\Rightarrow$ finds parametric lexicographically maximal element of a set bounded by linear constraints
Dataflow Analysis (2)

Static affine program

```c
for( j=1; j<=4; j++ ) {
    x[j] = F1( ... );
}
for( j=1; j<=4; j++ ) {
    if( j<=2 ) {
        x[j] = F2( x[j] );
    }
    ... = F3( x[j] );
}
```

Dependence analysis for \( F_3 \leftarrow F_2 \)

subject to:

\[
\begin{align*}
1 & \leq j_W \leq 4 \\
j_W & \leq 2 \\
F_2 & = j_R \\
1 & \leq j_R \leq 4
\end{align*}
\]

objective: \( j_W^{\text{max}} = \max_{\text{lex}} \{ j_W(j_R) \} \)

PIP Solution:

\[
\begin{align*}
\text{if } ( j_R \leq 2 ) & \text{ then } j_W = j_R \\
\text{else } & \text{ no solution;}
\end{align*}
\]

Meaning of the Solution:

if \( ( j_R \leq 2 ) \) then

\(<F_3 @ j_R > \text{ depends on } <F_2 @ j_W = j_R > \text{ via variable } x( j_W = j_R ) >

else

\(<F_3 @ j_R > \text{ depends on other functions }>

Dependencies expressed as PDG

Static affine program

```c
for( j=1; j<=4; j++ ) {
    x[j] = F1( ... );
}
for( j=1; j<=4; j++ ) {
    if( j<=2 ) {
        x[j] = F2( x[j] );
    }
    ... = F3( x[j] );
}
```

Polyhedral annotation:

\[ P_{F_2} = \{ j \in \mathbb{Z} \mid 1 \leq j \leq 2 \} = \{1,2\} \]
Embedded System-level Platform Synthesis and Application Mapping
**ESPAM**

**Embedded System-level Platform Synthesis and Application Mapping**

- Simple descriptions in XML format
- Automated System-to-RTL Level conversion and software code generation
- Ready for direct implementation
- Target: prototyping on FPGAs or desktop PCs
- All of this in a matter of hours
The MPSoC Architecture

Library of parameterized components:

- **Processing Components:**
  - Programmable processors
  - Hardware IP Cores

- **Memory Components:**
  - Program, Data (on-chip and external) Memory (MEM)
  - Communication Memory (CM)

- **Communication Components:**
  - Point-to-point network
  - Crossbar switch
  - Shared bus with Round-Robin, Fixed Priority, or TDMA arbitration

- **Communication Controller (CC)** – interface between processing, memory, and communication components

Many alternative platforms can be constructed fast and easily by instantiating different type/number of components and setting their parameters.
Communication and Synchronization

- Data communication and synchronization between processors only through FIFOs mapped in the Communication Memories (CM)
- A processor can write only to its local CM
- A processor can access other CMs only for read operations through the communication component using requests
- The synchronization mechanism is implemented by Read and Write SW primitives that interact directly with the HW Communication Controllers
Platform Synthesis and Programming

- Platform elaboration and Refinement
- Mapping of processes
- FIFOs to CMs mapping
- Memory map of the system
- Program code for each processor
- Read and write synchronization primitives

Polyhedral Process Network

Code of MB1

```c
int x=0;
for (int i=0; i<=N; i++) {
    execute_A( &x );
    writeCM( p1, x, 1 );
}
```

Code of MB2

```c
int y=0, z=0;
for (int i=0; i<=N; i++) {
    execute_B( y, &z );
    writeCM( p3, z, 1 );
}
```

Read/Write addresses of the FIFOs

```c
#define p1 0xe0000008 // write addr. FIFO1
#define p2 0x00010000 // read addr. FIFO1

#define p3 0xe0000008 // write addr. FIFO2
#define p4 0x00020000 // read addr. FIFO2
```
Write and Read to/from FIFOs

**Write Synchronization Primitive**

```c
#define writeCM( pos, value, n )
    int i;
    volatile int *isFull;
    volatile int *outPort=(volatile int *) pos;
    isFull = outPort + 1;
    for( i=0; i<n; i++ ) {
        while( *isFull ) { }
        *outPort = ((volatile int *) value)[i];
    }
```

**Read Synchronization Primitive**

```c
#define readCM( pos, value, n )
    int i;
    volatile int *isEmpty;
    int inPort = (int) pos;
    (volatile int *) Req = (volatile int *) 0xE0000000;
    isEmpty = Reg + 1;
    *Reg = 0x80000000 | inPort;
    for( i=0; i<n; i++ ) {
        while( *isEmpty ) { }
        ((volatile int *) value)[ i ] = *Reg;
    }
    *Reg = 0x7FFFFFFF & inPort;
```
Integration of dedicated HW IP cores
HW IP Core Integration with ESPAM

- To meet higher application requirements, we need to integrate hardware IP cores into ESPAM generated systems

- Easy and efficient integration of Hardware IPs with multiple processors in heterogeneous systems

- Features of generated HW Module with IP in it
  - modularity, i.e., HW Module consisting of well defined parameterized components
  - clearly defined interfaces between components of a HW module
  - loosely coupled components in a HW module
    - Easy to modify and optimize HW modules for better performance and area utilization
PPN to Heterogeneous MPSoC

- Processes mapped to programmable processors and/or dedicated HW IPs
- Bus, Crossbar, and Point-to-Point communication topologies supported
  - Communication Controllers
  - Communication Memories
  - FIFO components
- Hardware Modules – wrappers around predefined IP cores
- HW IPs must provide:
  - Function call behavior
  - Unidirectional I/O data interfaces
  - Enable/Valid control interface
Structure of a HW Module

- **Read Block**
  - Fetch data from communication channels
  - For each input argument select from which port to fetch the data using the control information derived from the PPN

- **Execute Block**
  - A functional sub-wrapper for the IP core

- **Write Block**
  - Write back the results from the execution to the communication channels
  - For each output argument select which port to receive the corresponding data

- **Control Block**
  - Control and synchronize reading, writing and execution

```c
1  // process P2
2  void main() {
3      for (int i=2; i<=N; i++)
4          for (int j=1; j<=M+i; j++) {
5              if (i-2 == 0)
6                  read(IP1, in_0, size);
7              if (i-3 >= 0)
8                  read(IP2, in_0, size);
9          execute(in_0, out_0);
10         if (-i+N-1 >= 0)
11             write(OP1, out_0, size);
12         if (i-N == 0) {
13             write(OP2, out_0, size);
14          } // for j
15      } // main
```
HW Module Structure in Detail

- Clearly structured and modularized
  - Library of predefined and parameterized components to build a HW module
  - Clearly defined interfaces between components
  - Simple HW module generation

- HW modules with loosely coupled components
  - Easy to debug and optimize components separately
  - Better performance and area utilization
Daedalus
A JPEG case study
Case study – JPEG encoder

Tile = 128 MacroBlocks

- Packet of bytes
- Compressed byte sequence for Tile

MacroBlock = 2Yblocks + 1Ublock + 1Vblock

Yblock = 64 pixels,
Ublock = 64 pixels,
Vblock = 64 pixels,
DSE Results
JPEG encoder MP-SoCs

Projected speed-ups

- Homogeneous MP-SoCs
- Heterogeneous MP-SoCs

<table>
<thead>
<tr>
<th>Number of processing cores in MP-SoC</th>
<th>Estimated speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>
Espam Synthesis Results (1)

JPEG case study, homogeneous systems (32 tiles):
Espam Synthesis Results (1)

JPEG case study, heterogeneous systems (32 tiles):
To Summarize

The best of all, we performed

✓ The DSE study (≤ 5% error) and
✓ The implementation of 25 MP-SoC JPEG encoder variations on an FPGA in only 5 days!
✓ Combining data and task parallelism
  24 cores, 19.7x speed-up, 288KB memory
Merits of the Daedalus design-flow

- Automated parallelization of media/streaming applications into parallel specifications (PPNs)

- Automated synthesis of MP-SoC platforms at system level, in a plug-and-play fashion

- Automated mapping of parallel application specifications onto MP-SoC platforms

- Results as good, as good are the library components
Making system-level design take off

Universiteit Leiden

http://daedalus.liacs.nl