A Model-based Embedded SW Development Methodology for MPSoC

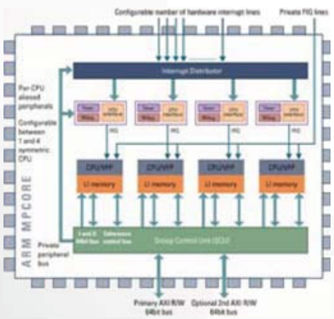
June 25, 2007

Soonhoi Ha
Seoul National University

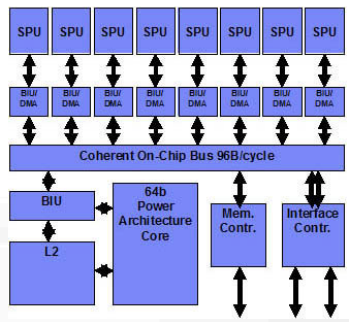
Contents

Technology Trend

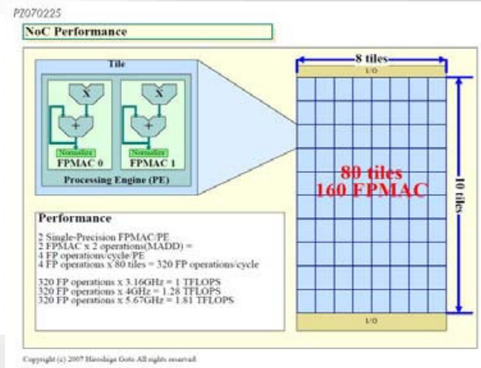
- More processor cores in a single chip
  - MPSoC (Multiprocessor System on Chip)
  - from multicore to manycore
    - UCB prediction: “The Landscape of Parallel Computing Research: A view from Berkeley,” Technical report, Dec. 18, 2006: 1000 processor cores with 30nm process
  - Heterogeneous architectures



ARM MPCore



IBM Cell



Motivation

- Platform based design
  - Platform: common (HW/SW) denominator over multiple applications
  - Pre-built and verified (HW/SW) architecture
  - SW design and system verification are major challenges

- Current SoC-related projects (in Korea) focus on hardware design and verification

- Software design on MPSoC
  - Embedded software with timing and resource constraints
  - Parallel programming for heterogeneous multiprocessors
  - Fast virtual prototyping

Current Practice (in Korea)

- Virtual prototyping
  - Coware ConvergenSC, ARM(Axys) MaxSim
  - Manual software and hardware partitioning
  - TLM (transaction level modeling) simulation
    - Simulation speed becomes important as the number of processors increases. (4 processors – under 100 Kcycles/sec)
  - Software debugging capability is limited
- S/W programming
  - Mainly manual design
  - Start considering UML based tool: (ex) Telelogic TAU
    - Limited capability: documentation, code structure

Problem Statement

- Target: MPSoC with high degree of parallelism
  - Scalability
  - Heterogeneous processors with diverse communication architecture
  - Power-constrained system
- Problem: parallel programming for MPSoC
  - Parallelism extraction (multiple use case, multi-tasking apps.)
    - Functional parallelism, data-parallelism, temporal-parallelism
  - Partitioning and mapping
  - Parallel code generation: parallel programming is not easy
  - Performance estimation and verification
  - Design space exploration

→ Need of sound (scalable and robust) methodology

Our Related Work: PeaCE

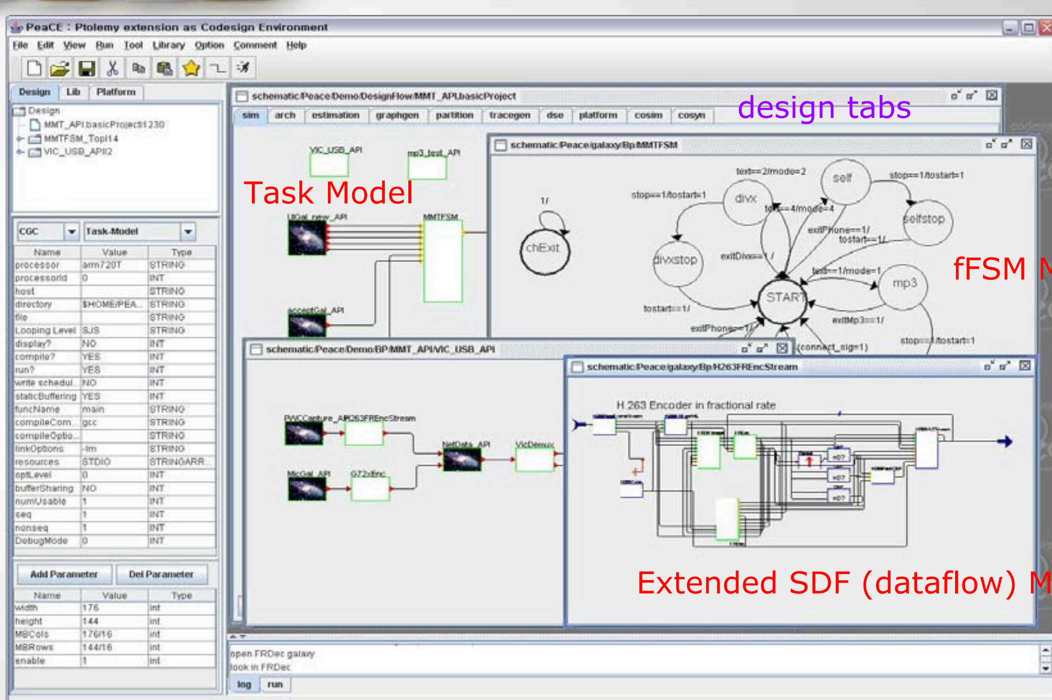
PeaCE

- Full-fledged codesign environment from system level specification to system synthesis
- Codesign environment being developed by CAP Laboratory, Seoul National University, Korea
- Built on top of Ptolemy classic of U.C.Berkeley (ptolemy.eecs.berkeley.edu)
- Open source program for academic purpose
- Exhibited at the University Booth in DAC 2003-2006

PeaCE home page

- http://peace.snu.ac.kr/research/peace
- papers, manual, tutorial materials, etc

PeaCE Design Environment



We Need More than PeaCE

- Parallelism
  - PeaCE focuses on function parallelism: task model, dataflow model
  - How to extract data parallelism and temporal parallelism?
- Decoupled specification and implementation
  - PeaCE: tight coupling of model and implementation
  - Allow diverse pairs of {model, implementation}
- Diverse communication architecture
  - PeaCE: consider shared-bus architecture only
  - Design space exploration covering bus-matrix and NoC
- Virtual prototyping
  - PeaCE: virtual synchronization achieves good performance
  - How to improve the cosimulation performance further? Parallel cosimulation

Related Work (1)

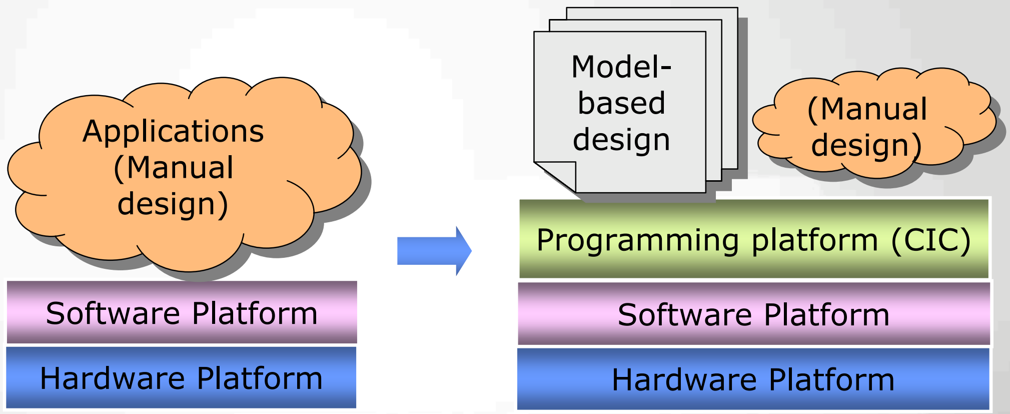
- Model-based design of embedded SW
  - UML-based tool: IBM Rose RT, Telelogics TAU (iLogix Rapsody): control-oriented application, targetting single processor system
  - SIMULINK-based embedded SW generation: limited success for automotive application
  - UC Berkeley: Metropolis project (A. Sangiovanni-Vincentelli)
  - Royal Institute of Tech. Sweden: ForSyDe (Axel Jantsch)
- Parallel Programming Environment
  - MPI, OpenMP: targetting general purpose processors
    - MPI: distributed memory system
    - OpenMP: shared address space system

Related Work (2)

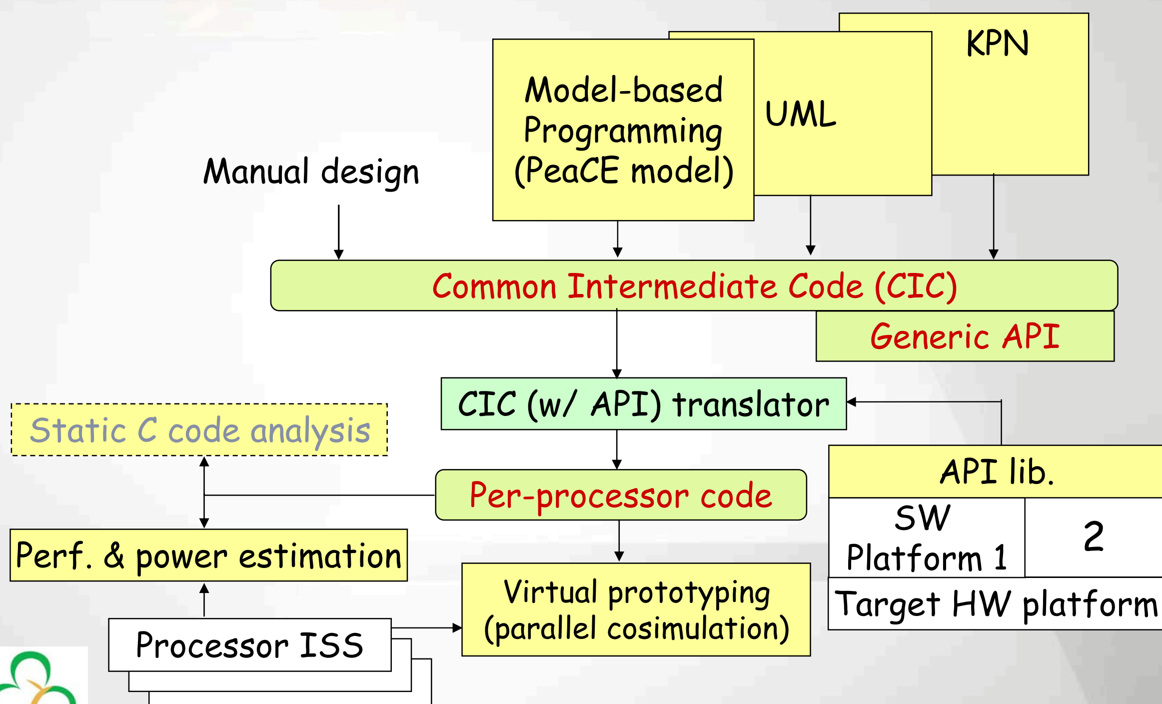
- Virtual prototyping system
  - TLM simulation: Synopsys CCSS, Coware ConvergenSC, ARM MaxSim: focusing on hardware/software co-validation. Not easy to debug parallel program
  - Lock-step synchronization: poorer performance as more cores are integrated
- Communication architecture optimization
  - NoC optimization: Xpipe, OCCN, etc.
- Partitioning and mapping
  - Single task application and/or functional parallelism

Contents

Basic Idea



HOPES Proposal



Key Techniques

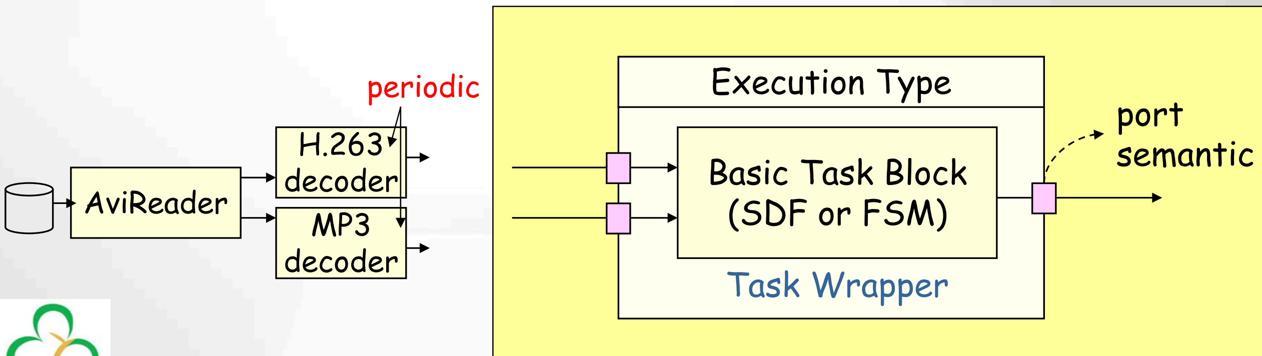
- SW design techniques
  - Model-based specification
  - Partitioning and automatic code generation from specification
  - Parallel programming (task parallelism, data parallelism)
  - Generic APIs (independent of OS and target architecture)
- SW verification techniques: 3-phase verification
  - At specification level: static analysis of program specification and functional simulation
  - After code generation (optional): static analysis of C code
  - At simulation time: debugging on virtual prototyping environment

PeaCE Model

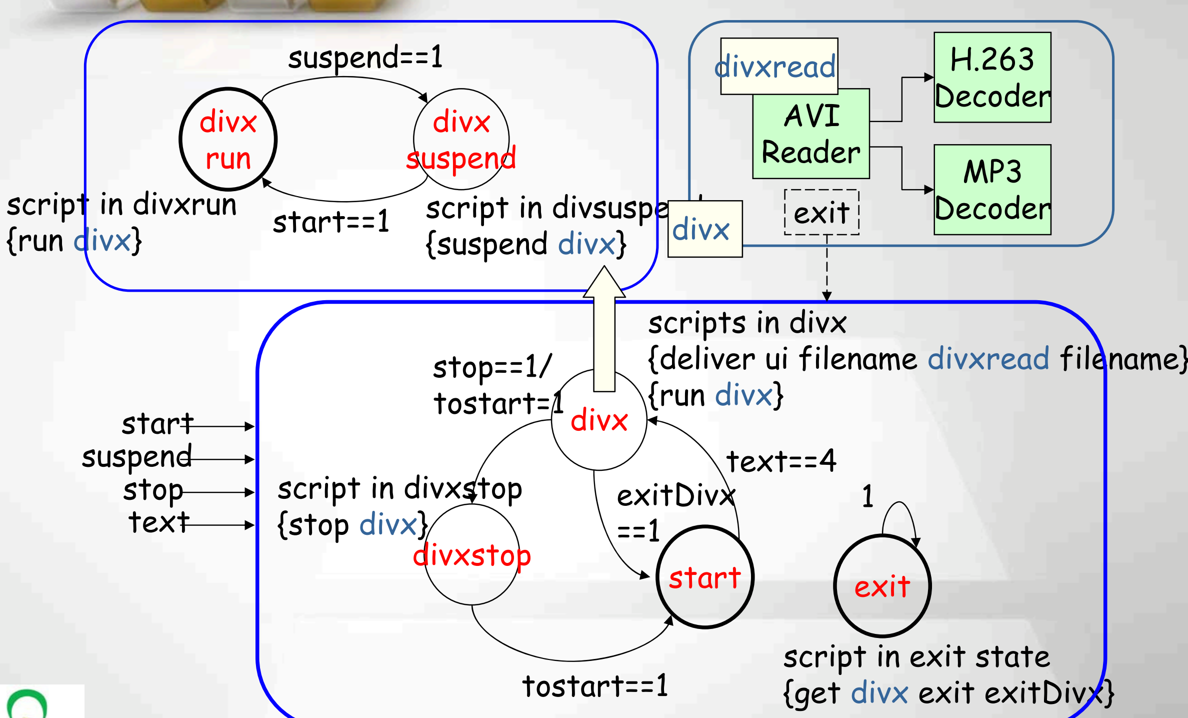
- Top model: Task model
  - Multi-mode multi-task application
- Computation task model: Dataflow model
  - Extended SDF model: SPDF (Synchronous Piggybacked Data Flow) and FRDF (Fractional Rate Data Flow)
  - A node represents a function block: naturally express functional parallelism
- Control task model: FSM model
  - Hierarchical and concurrent FSM model: fFSM (flexible Finite State Machine)
  - Communication with SPDF model: script language

Task Model

- Task execution semantics
  - periodic, sporadic, functional
- Port properties
  - data rate : static or dynamic
  - data size : static or variable
  - port type : queue or buffer



fFSM + SPDF



Contents

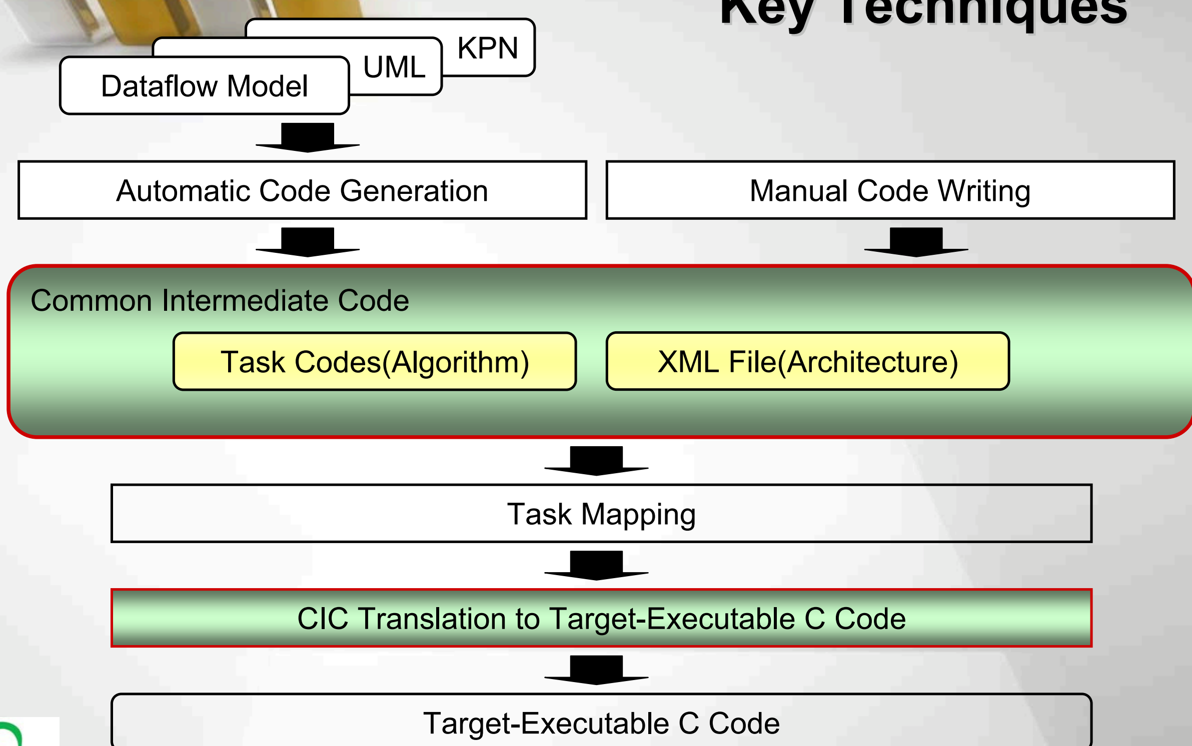
1. Introduction

2. Proposed Design Flow

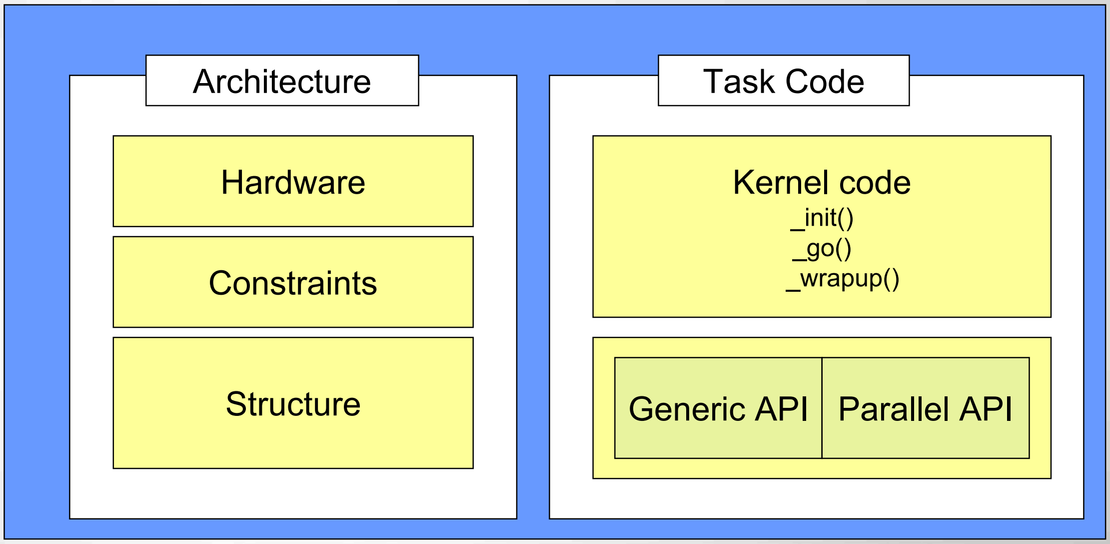
3. Key Techniques
   - CIC: Common Intermediate Code
   - CIC translator

4. Preliminary Experiments

Key Techniques



CIC Format

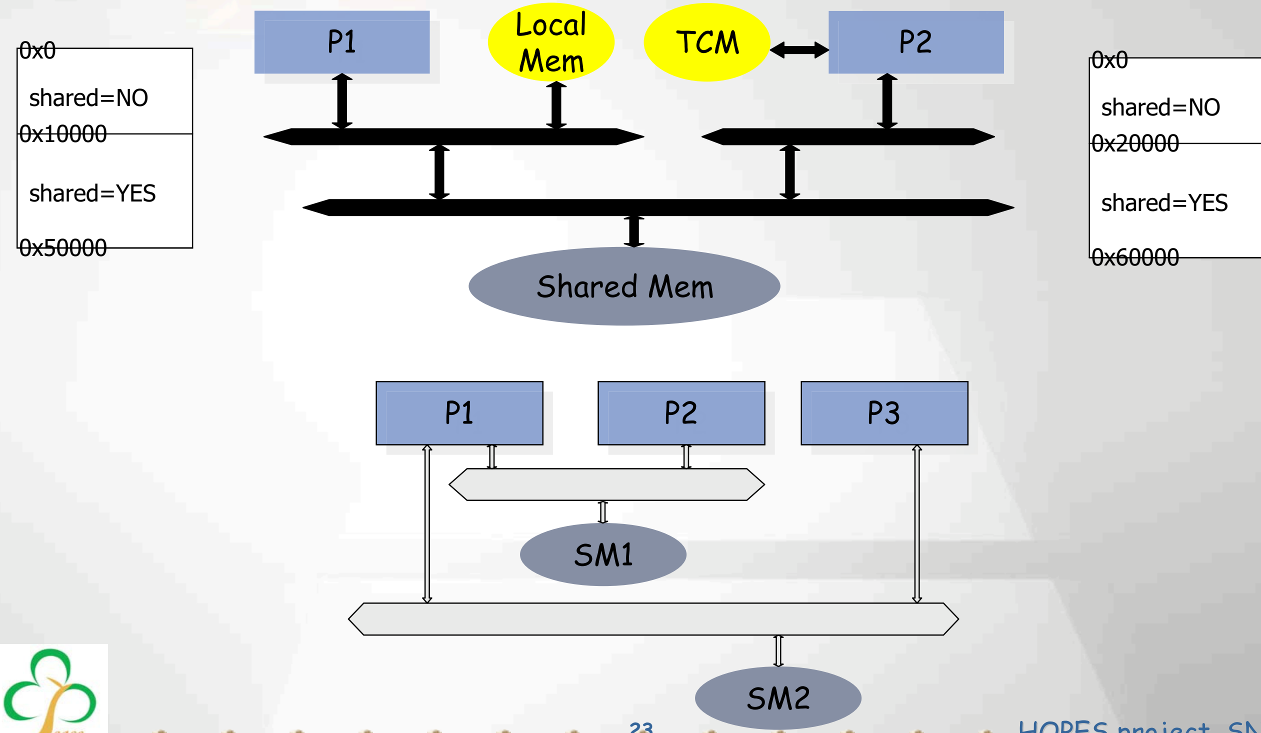


DivX player CIC xml (Hardware)

```
<?xml version="1.0" ?>
 <CIC_XML>
  <hardware>
    <processor name="arm926ej-s0">
      <index>0</index>
      <localMem name="lmap0">
        <addr>0x0</addr>
        <size>0x10000</size>  // 64KB
      </localMem>
      <sharedMem name="shmap0">
        <addr>0x10000</size>
        <size>0x40000</size>  // 256KB
        <sharedWith>1</sharedWith>
      </sharedMem>
      <OS>
        <support>TRUE</support>
      </OS>
    </processor>
```

```
    <processor name="arm926ej-s1">
      <index>0</index>
      <localMem name="lmap1">
        <addr>0x0</addr>
        <size>0x20000</size>  // 128KB
      </localMem>
      <sharedMem name="shmap1">
        <addr>0x20000</size>
        <size>0x40000</size>  // 256KB
        <sharedWith>0</sharedWith>
      </sharedMem>
      <OS>
        <support>TRUE</support>
      </OS>
    </processor>
  </hardware>
```

Hardware Architectures



DivX player CIC xml (Constraint)

```
<constraints>
  <memory>16MB</memory>
  <power>50mWatt</power>
    <mode name="default">
    <task name="AviReaderI0">
      <period>120000</period>
      <deadline>120000</deadline>
      <priority>0</priority>
        <subtask name="arm926ej-s0">
        <execTime>186</execTime>
        </subtask>
      </task>
      <task name="H263FRDivxI3">
      <period>120000</period>
      <deadline>120000</deadline>
      <priority>0</priority>
        <subtask name="arm926ej-s0">
        <execTime>5035</execTime>
        </subtask>
      </task>
      <task name="MADStreamI5">
      <period>120000</period>
      <deadline>120000</deadline>
      <priority>0</priority>
        <subtask name="arm926ej-s0">
        <execTime>13</execTime>
        </subtask>
      </task>
    </mode>
</constraints>
```

DivX player CIC xml (Structure)

```
<structure>
 <mode name="default">
  <task name="AviReaderI0">
   <subtask name="arm926ej-s0">
   <procMap>0</procMap>
   <fileName>AviReaderI0_arm926ej_s0.cic</fileName>
   </subtask>
  </task>
  <task name="H263FRDivxI3">
   <subtask name="arm926ej-s0">
   <procMap>0</procMap>
   <fileName>H263FRDivxI3_arm926ej_s0.cic</fileName>
   </subtask>
  </task>
  <task name="MADStreamI5">
   <subtask name="arm926ej-s0">
   <procMap>0</procMap>
   <fileName>MADStreamI5_arm926ej_s0.cic</fileName>
   </subtask>
  </task>
 </mode>
```

```
  <queue>
   <name>mq0</name>
   <src>AviReaderI0</src>
   <dst>H263FRDivxI3</dst>
   <size>30000</size>
  </queue>
  <queue>
   <name>mq1</name>
   <src>AviReaderI0</src>
   <dst>MADStreamI5</dst>
   <size>30000</size>
  </queue>
</structure>
</CIC_XML>
```


Task Code: Structure

- Task kernel code
  - _init(): before main loop
  - _go(): in the main loop
  - _wrapup(): after main loop

```
(example) h263dec.cic
// header file
// global declaration and definition
// procedure definition
    h263dec_init() { … }
    h263dec_go() { … }
    h263dec_wrapup() { … }
```

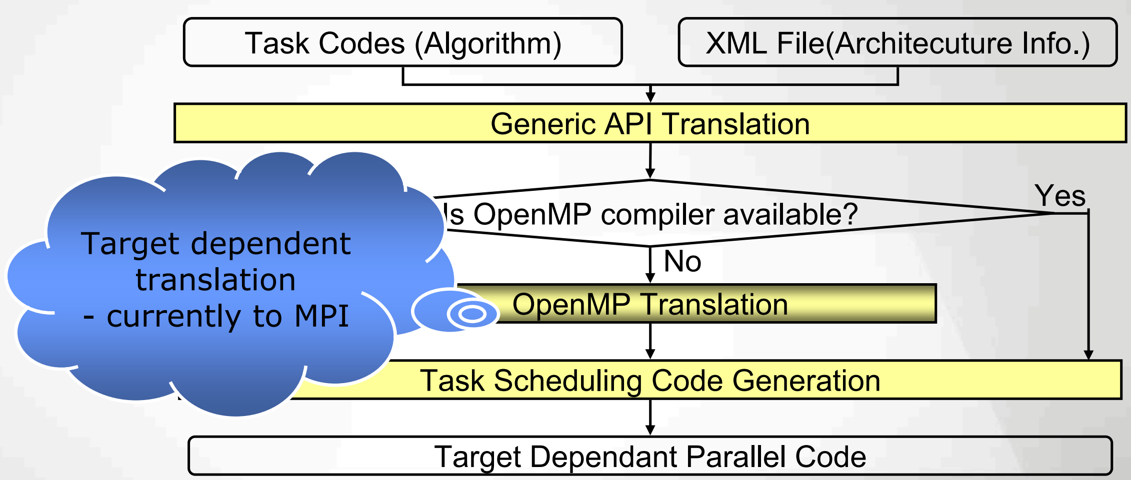
```
// Task Initialize
Task_init();

// Main Body
while(1) {
  Task_go();
}

// Finishing
Task_wrapup();
```

CIC Code Generation

- Separate CIC task codes for partitioned tasks
  - Explicit functional parallelism → task_name.cic
- openMP programming for data parallelism
- Generic API for platform independent programming

```
void h263decoder_go (void) {
        ...
        l = MQ_RECEIVE("mq0", (char *)(ld_106->rdbfr), 2048);
        ...
        # pragma omp parallel for
        for(i=0; i<99; i++) {
             //thread_main()
           ....
        }
         // display the decode frame
           dither(frame);
}
```


CIC Translator



Generic API

Generic API

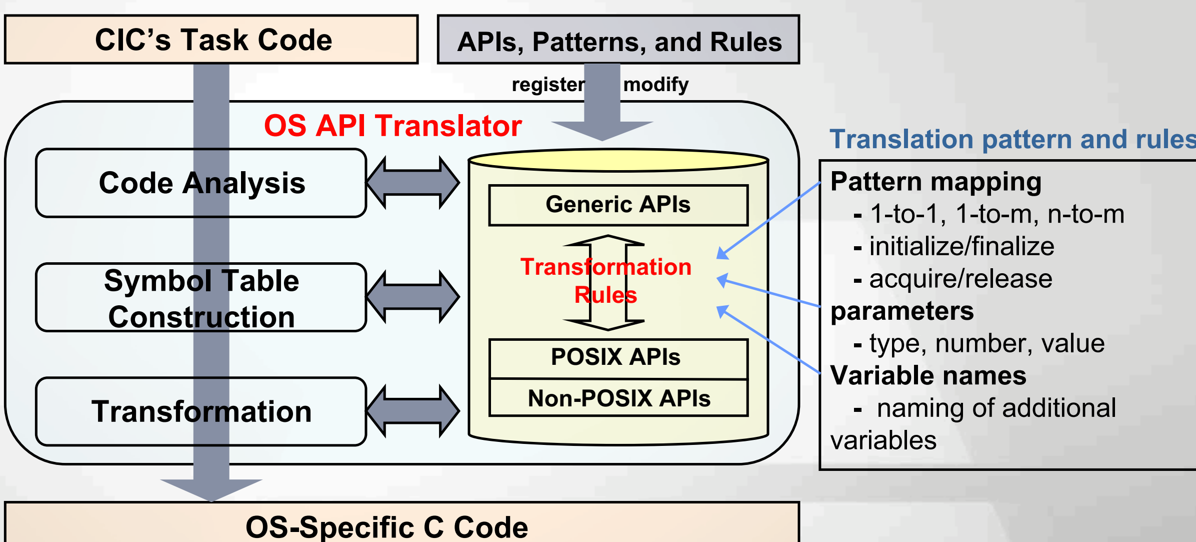
- OS-indepednent API
- Abstraction from IEEE POSIX 1003.1-2004
- Selected OS services + C library functions

```
file = OPEN("input.dat", O_RDONLY);
…
READ(file, data, 100);
…
CLOSE(file);
```

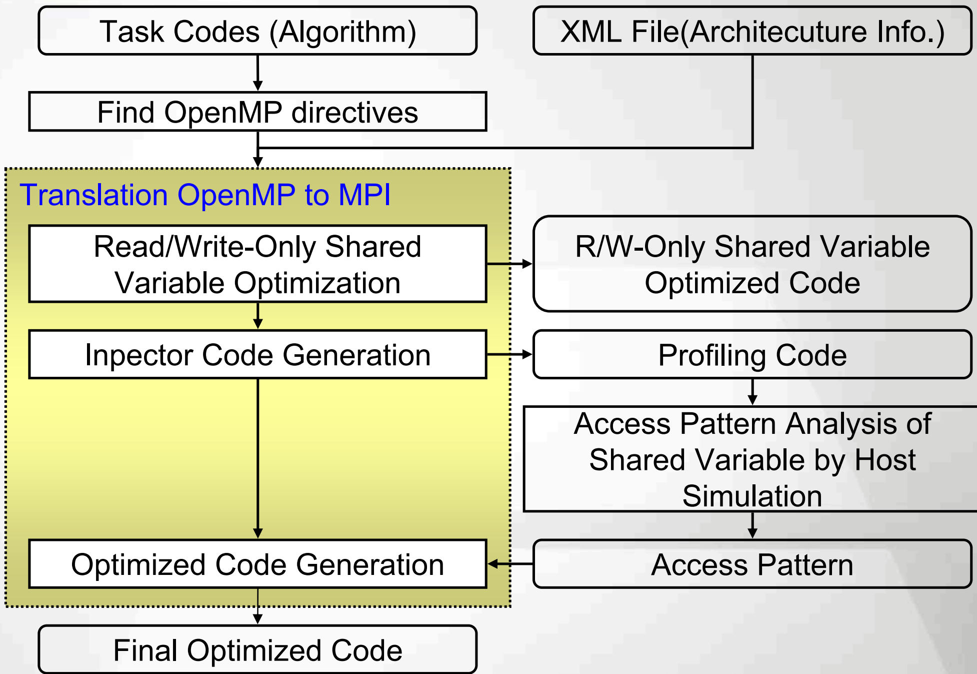
```
#include <stdio.h>
…
file = fopen("input.dat", "r");
…
fread(data, 1, 100, file);
…
fclose(file);
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
…
file = open("input.dat", O_RDONLY);
…
read(file, data, 100);
…
close(file);
```


Internal Structure of API Translator



OpenMP to MPI Translator



Naive Translation

```
/* Simple example of an OpenMP program */
int main() {
  int i;
  int a[1000];
  int b[1000];
#pragma omp parallel shared(a,b) private(i)
#pragma omp_hopes read_only(a) write_only(b)
 {
#pragma omp for
   for(i = 0; i < 1000; i++)
      b[i] = a[i] + i;
 }
}
```

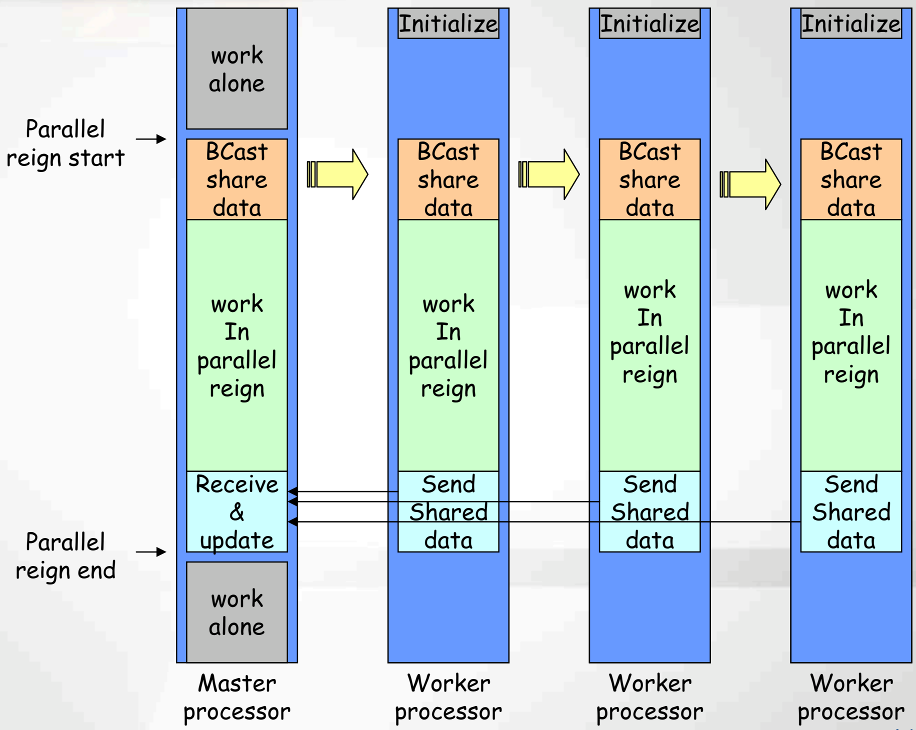
master

```
/* Simple example of an OpenMP program */
int main() {
  int i;
  int a[1000], b[1000];
  {
    int rank,numtasks;
    int argv = 0;
    MPI_Init(argv);
    …
    MPI_Bcast(&a,1000,MPI_INT,0,MPI_COMM_WORLD);
    MPI_Bcast(&b,1000,MPI_INT,0,MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);
    {
      …
      int  temp_a[1000];
      int  temp_b[1000];
      int  other_temp_a[1000];
      int  other_temp_b[1000];
      1) save initial data of shared variables in temporary variables
      for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)
          (*((b)+(i)))=((*((a)+(i)))+(i)));
      2) update result data into each shared variable
      for(_temp_I=0;_temp_I<(2-1); _temp_I++){
        MPI_Recv(&other_temp_a,1000,MPI_INT,_temp_I+1,
             0,MPI_COMM_WORLD,&stat);
        MPI_Recv(&other_temp_b,1000,MPI_INT,_temp_I+1,
             1,MPI_COMM_WORLD,&stat);
     3) compare received data with initial data, and update modified value
      }
      MPI_Barrier(MPI_COMM_WORLD);
      MPI_Finalize();
  }
}
```

slave

```
void worker_1(int argv) {
  int a[1000], b[1000];
  int i;
  …
  MPI_Init(argv);
  ..
  MPI_Bcast(&a,1000,MPI_INT,0,MPI_COMM_WORLD);
  MPI_Bcast(&b,1000,MPI_INT,0,MPI_COMM_WORLD);
  MPI_Barrier(MPI_COMM_WORLD);
 {
    for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*
        (rank+1))/2)+0;(i)++)  (*((b)+(i)))=((*((a)+(i)))+(i)));
    MPI_Send(&a,1000,MPI_INT,0,0,MPI_COMM_WORLD);
    MPI_Send(&b,1000,MPI_INT,0,1,MPI_COMM_WORLD);
 }
 MPI_Barrier(MPI_COMM_WORLD);
}
```

Worker model of execution



Read/write-only Variable Optimization

Master

```
1-9      …
10-11    MPI_Bcast(&a,1000,MPI_INT,0,MPI_COMM_WORLD);
12       MPI_Barrier(MPI_COMM_WORLD);
13       {
14         …
15-16      int  temp_b[1000];
17-18      int  other_temp_b[1000];
19         1) save only b's initial value in temporary variables
20         for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)    (*((b)+(i)))=((*((a)+(i)))+(i)));
22         2) update result of b
23         for(_temp_l=0;_temp_l<(2-1); _temp_l++){
24-25          MPI_Recv(&other_temp_b,1000,MPI_INT,_temp_l+1, 1,MPI_COMM_WORLD,&stat);
26         3) compare received data with initial data and update modified one
27       }
28-31    …
```

Worker

```
1-6     …
7-8     MPI_Bcast(&a,1000,MPI_INT,0,MPI_COMM_WORLD);
9       MPI_Barrier(MPI_COMM_WORLD);
10      {
11         for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)
12            (*((b)+(i)))=((*((a)+(i)))+(i)));
13-14      MPI_Send(&b,1000,MPI_INT,0,1,MPI_COMM_WORLD);
15      }
16-17   …
```

Inspector Code Optimization(1)

- Inspector code generation

Master

```
1-9     …
        MPI_Barrier(MPI_COMM_WORLD);
        {
            ...
            master_inspector((void*)a,2,1,sizeof(int ),0);
            master_inspector((void*)b,2,2,sizeof(int ),0);
        }
12      MPI_Barrier(MPI_COMM_WORLD);
20      for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*
21          (rank+1))/2)+0;(i)++)  (*((b)+(i)))=((*((a)+(i)))+(i)));
        master_inspector((void*)a,2,1,sizeof(int ),1);
        master_inspector((void*)b,2,2,sizeof(int ),1);
28-31   …
```

Worker

```
1-6     …
9       MPI_Barrier(MPI_COMM_WORLD);
        {
          ...
          int t_a[1000], t_b[1000];
          initialize_table(t_a,1000);
          initialize_table(t_b,1000);
          for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)
          {
             inspector(t_a,(void*)a,(void*)(a+i),sizeof(int ));
             inspector(t_b,(void*)b,(void*)(b+i),sizeof(int ));
          }
          recv_initial_data(t_a,(void*)a,1000,sizeof(int ),1,rank,0);
          recv_initial_data(t_b,(void*)b,1000,sizeof(int ),2,rank,0);
          MPI_Barrier(MPI_COMM_WORLD);
11        for((i)=(((1000-0)*(rank))/2)+0;(i)<(((1000-0)*(rank+1))/2)+0;(i)++)
12           (*((b)+(i)))=((*((a)+(i)))+(i)));
          recv_initial_data(t_a,(void*)a,1000,sizeof(int ),1,rank,1);
          recv_initial_data(t_b,(void*)b,1000,sizeof(int ),2,rank,1);
        }
16-17 …
```


Inspector Code Optimization(2)

- Optimized Code

Master

```
1-9     …
12      MPI_Barrier(MPI_COMM_WORLD);
13      {
14      ...
        MPI_Send(&a[500],500,MPI_INT,1,0,MPI_COMM_WORLD);
20      for((i)=(((1000-0)*(rank))/4)+0;(i)<(((1000-0)*(rank+1))/4)+0;(i)++)
21         (*((b)+(i)))=((*((a)+(i)))+(i)));
23-26   MPI_Recv(&a[500],500,MPI_INT,1,0,MPI_COMM_WORLD,&stat);
27      }
28-31 …
```

Worker

```
1-6     …
9       MPI_Barrier(MPI_COMM_WORLD);
10      {
          ...
          MPI_Recv(&a[500],500,MPI_INT,0,0,MPI_COMM_WORLD,&stat);
11        for((i)=(((1000-0)*(rank))/4)+0;(i)<(((1000-0)*(rank+1))/4)+0;(i)++)
12           (*((b)+(i)))=((*((a)+(i)))+(i)));
13-14     MPI_Send(&b,1000,MPI_INT,0,1,MPI_COMM_WORLD);
15      }
16-17 …
```

Scheduling Code (1): w/o OS

```
typedef struct {
  void (*init)();
  int (*go());
  void (*wrapup)();
  int period;
  int time_count;  /* this variable indicates the next invocation time of a task */
} task;
int taskNum = 2;
task taskInfo[] = { {task1_init, task1_go, task1_wrapup, 100, 0}
    , {task2_init, task2_go, task2_wrapup, 200, 0}};
…
int main() {
  init();            /* {task_name}_init() functions of all tasks are called */
  scheduler(); /* scheduler code */
  wrapup();     /* {task_name}_wrapup() functions of all tasks are called */
  return 0;
}
```


Scheduling Code (2): w/o OS

```
void scheduler() {
  while(all_task_done()==FALSE) {
    int taskId = get_next_task();
    if (taskInfo[taskId]->go()==0) {
      taskInfo[taskId]->timeCount += taskInfo[taskId]->period;
    }
  }
}
/* return the number of task id that should be executed */
int get_next_task() {
  for all task in this processor
    find the tasks that has the smallest value of time_count variable
  if (the number of found tasks > 1) {
    for found tasks that has the smallest value of time_count variable
      select the task that is not executed for the longest time in found tasks
  }
  return task_id;
}
```

Scheduling Code (3): w/ OS

```
void *thread_task_0_func (void *argv) {
  …
  task_0_go();
  get_time(&time);
  sleep(task_0->next_period – time);    //sleep during remained time
  …
}
int main() {
  …
  pthread_t thread_task_0;
  sched_param thread_task_0_param;
  …
  thread_task_0_param.sched_priority = 0;
  pthread_attr_setschedparam (…, &thread_task_0_param);
  …
  task_init();   /* In this function, {task_name}_init() of each is called  */
  pthread_create (&thread_task_0, &thread_task_0_attr, thread_task_0_func, NULL);
  …
  task_wrapup();   /* In this function, {task_name}_wrapup() of each task is called */
}
```

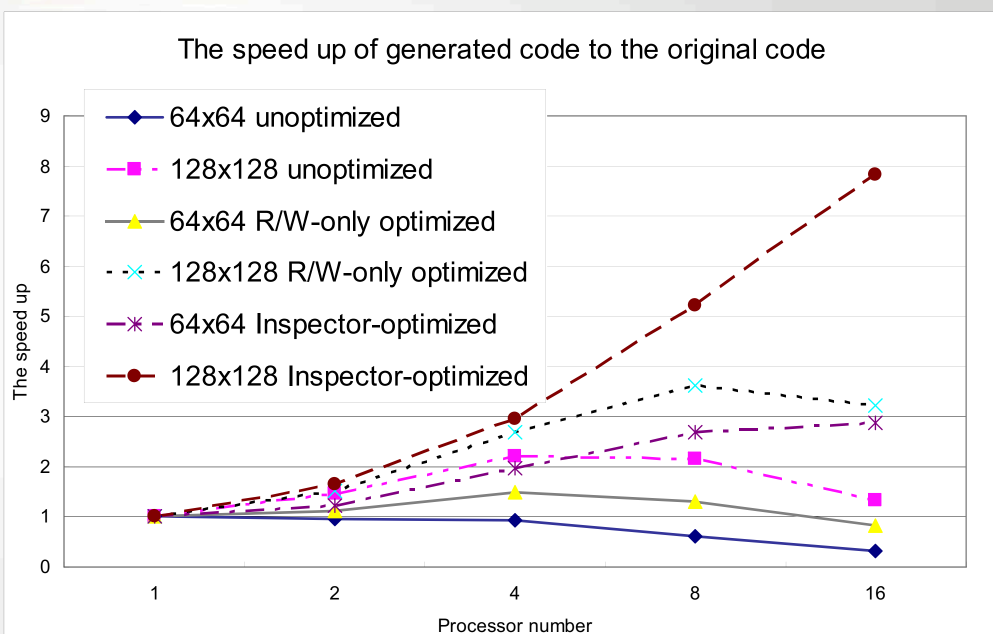

Contents

Matrix Multiplication

- OpenMP translator

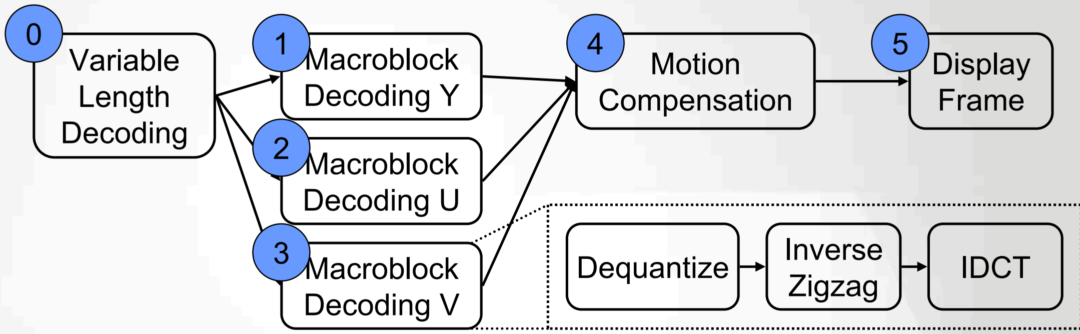
```
int matA[SIZE][SIZE], matB[SIZE][SIZE], matC[SIZE][SIZE];
...
int i, j, k;
#pragma omp parallel private(i,j,k) shared(matC, matA, matB)
#pragma omp_hopes read_only(matA, matB) write_only(matC)
{
  #pragma omp for   for (i = 0; i < SIZE; i++)
     for (j = 0; j < SIZE; j++)
        for (k = 0; k < SIZE; k++)
            matC[i][j] += matA[i][k] * matB[k][j];
}
```


Matrix Multiplication (2)



H.263 Decoder

- Task graph and partitioning



Configurations and Execution Cyles

- Task mapping

Processor id	Configuration (task mapping)		
	1	2	3
0	Task 0, 1, 2, 3, 4, 5	Task 0, 2, 3, 4, 5	Task 0, 3, 4, 5
1	N/A	Task 1	Task 1
2	N/A	N/A	Task 2

- Execution Cycles

# processors for data parallelism	Configuration (task mapping)		
	1	2	3
No OpenMP	158,099,172	146,464,503	146,557,779
2	167,119,458	152,753,214	153,127,710
4	168,640,527	154,159,995	155,415,942

Comparison with Manual Coding

- Number of code lines for functional parallelism

	Algorithm	Communication	Total
Code for processor 1	2393	226	2619
Code for processor 1	400	226	626
Original code			2507

- Number of code lines for data parallelism

	Data parallel region		
	Original	Master	Slave
Number of code line	56	97	249
Time to write	12 man hours		

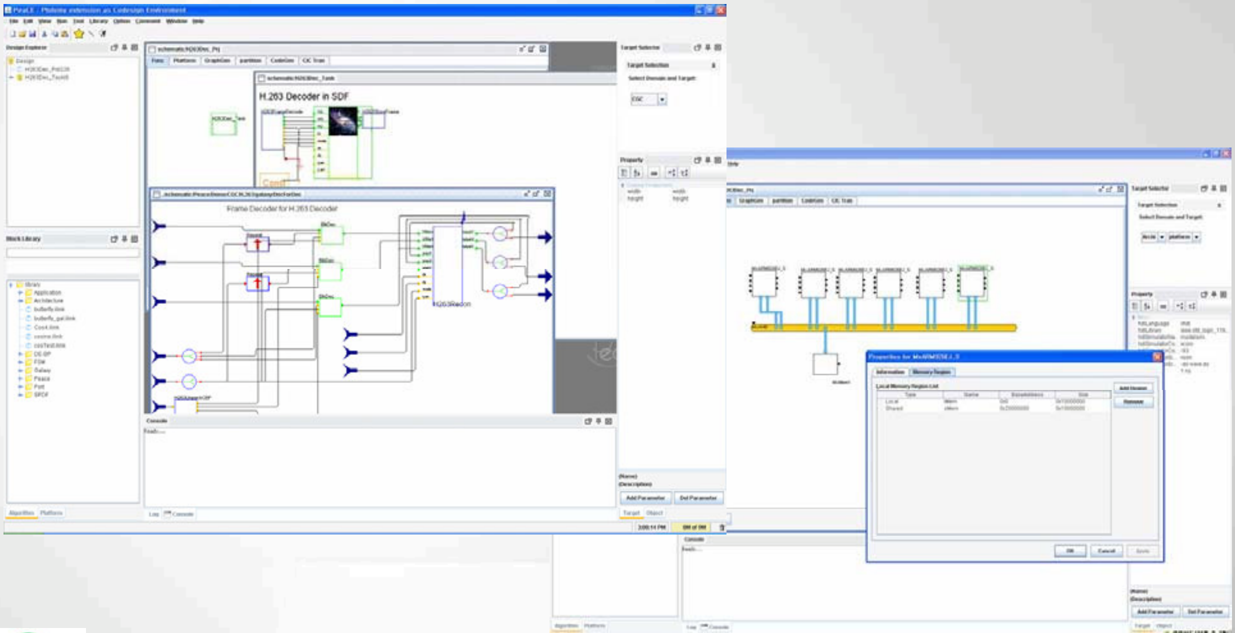
- Performance comparison with manual coding
  - Manual: 143,459,583 cycles,   Auto: 158,099,172 cycles
  - Overhead: 10.2 %

Status

- Project 1
  - Title: Development of Embedded software design and verification techniques for MPSoC,
  - Period: 2005.3.1 - 2007.2.28,
  - Amount: $2,800,000
  - Sponsor: Korean Ministry of Information and Communication

- Project 2
  - Title: New design methodology of next-generation embedded systems with high degree of parallelism
  - Period: 2007.5.1 – 2010.2.28
  - Amount: $520,000
  - Sponsor: Korean Ministry of Science and Technology

Demonstration



Conclusion

- HOPES is a newly launched project to make a embedded software development environment for MPSoCs
  - Support of diverse models
  - Target independent environment + target specific libraries
  - 3-phase verification: model-level, C code static analysis, run-time simulation
  - Integration of software modules at various stages
  - http://peace.snu.ac.kr/hopes (sorry, only Korean for now. English homepage will be open soon)