TDDI08: Embedded Systems Design
- Laboratory introduction -

Adrian Lifa
Embedded Systems Laboratory
Dept. of Computer and Information Science
Linköpings universitet

www.ida.liu.se/~adrli
adrian.alin.lifa@liu.se
Building B, Room 329:206
Lab 2: Design space exploration in MPARM
Design space exploration with MPARM

- HW platform
- SW platform
- Design space exploration for energy minimization
- Communication in MPSoC
MPSoC Architecture

ARM
CACHE

ARM
CACHE

ARM
CACHE

Interrupt Device

Private Memory

Private Memory

Private Memory

Semaphore Device

Shared Memory

Bus
System Design Flow

Hardware platform

Software Application(s)

Extract Task Graph

Extract Task Parameters

Optimize (Mapping & Sched)

Implement

Formal Simulation
MPARM: Hardware

- Simulation platform
- Up to 8 ARM7 processors
- Variable frequency (dynamic and static)
- Split/Unified instruction and data caches
- Private memory for each processor
- Scratchpad
- Shared Memory
- Bus: AMBA

- This platform can be fine tuned for a given software application

- Read more in:
  - /home/TDTS07/sw/mparm/MPARM/doc/simulator_statistics.txt
- C crosscompiler chain for building the software applications
- No operating system
  - Small set of functions (pr, shared_alloc, WAIT/SIGNAL)
  - (look in the application code)
MPARM: Why?

- Cycle accurate simulation of the system
- Various interesting statistics: number of clock cycles executed, bus utilization, cache efficiency, energy/power consumption of the components (CPU cores, bus, memories)
MPARM: How?

- `mpsim.x -c2` runs a simulation on 2 processors, collecting the default statistics
- `mpsim.x -c2 -w` runs a simulation with power/energy statistics
- `mpsim.x -c1 --is=9 --ds=10`: one processor with instruction cache of size 512 and data cache of 1024 bytes
- `mpsim.x -c2 -F0,2 -F1,1 -F2,3`: two processors running at 100 MHz, 200 MHz and the bus running at 66 MHz
  - 200 MHz is the “default” frequency
- Dynamic frequency scaling is possible (see the lab manual for instructions on how to modify the source code)
- `mpsim.x -h` for the rest

- Simulation results are in the file `stats.txt`
Design Space Exploration

- Generic term for system optimization
- **Platform optimization:**
  - Select the number of processors
  - Select the speed of each processor
  - Select the type, associativity and size of the cache
  - Select the bus type
- **Application optimization**
  - Select the interprocessor communication style
    (shared memory or distributed message passing)
  - Select the best mapping and schedule
Assignment 1

- Given the GSM code
- Running on 1 ARM7 processor
- The variables are:
  - cache parameters
  - processor frequency
- Using the MPARM simulator, find a hardware configuration that minimizes the energy of the system
Energy/Speed Tradeoff

CPU model

- **RUN**
  - 0.75V, 60mW, 150MHz
  - 1.3V, 450mW, 600MHz
  - 1.6V, 900mW, 800MHz

- **IDLE**
  - 40mW

- **SLEEP**
  - 160μW

Transition times:
- 160μs
- 10μs
- 10μs
- 140ms
- 1.5ms
- 90μs
Frequency Selection: ARM Core Energy
Frequency Selection: Total Energy

![Graph showing total energy expenditure as a function of frequency divider, with energy on the y-axis and frequency divider on the x-axis. The graph shows a general trend of increasing energy with increasing frequency divider.]
Instruction Cache Size: Total Energy

![Graph showing energy as a function of log2(Cache Size)]

- **Energy [mJ]**
  - 12.5
  - 12.0
  - 11.5
  - 11.0
  - 10.5
  - 10.0
  - 9.5
  - 9.0
  - 8.5

- **Cache Size**
  - $2^9 = 512$ bytes
  - $2^{14} = 16$ kbytes

The graph shows the total energy consumption as the cache size increases, with energy decreasing as the log2 of cache size grows, reaching a minimum around $2^{11}$, before increasing again at $2^{14}$. This indicates an optimal cache size for energy efficiency around 2048 bytes.
Instruction Cache Size: Execution Time

![Graph showing the relationship between log2(CacheSize) and execution time (t [cycles]).](image_url)

- Execution time decreases as the cache size increases.
- The graph indicates an exponential relationship between cache size and execution time.
Interprocessor Data Communication

CPU\(_1\)

... 

\textbf{a}=1 

...

CPU\(_2\)

... 

print \textbf{a};

...

BUS

How?
Shared Memory

CPU\(_1\)
...
\(a=1\)
...

CPU\(_2\)
\(a=2\)
print \(a\);

Shared Mem
\(a\)

BUS

Synchronization
Synchronization

With semaphores

CPU_1
a=1
signal(sem_a)

Semaphore
sem_a

CPU_2
wait(sem_a)
a=2
print a;

Shared Mem
a

BUS

a=2
Synchronization In[f/t]ernals

CPU_1
\[ a = 1 \]
\[ \text{signal(sem}_a \text{)} \]

sem_a = 1

Semaphore
\[ \text{sem}_a \]

CPU_2
\[ \text{while (sem}_a \text{==0)} \]
\[ \text{wait(sem}_a \text{)} \]
\[ a = 2 \]
\[ \text{print a;} \]
Disadvantages of polling:

- Results in higher power consumption (energy from the battery)
- Larger execution time of the application
- Blocking important communication on the bus
Instead:

- Direct CPU-CPU communication with distributed semaphores
- Each CPU has its own scratchpad
  - Smaller and faster than a RAM
  - Smaller energy consumption than a cache
  - Put frequently used variables on the scratchpad
  - Cache controlled by hardware (cache lines, hits/misses, ...)
  - Scratchpad controlled by software (e.g., compiler)

- Semaphores allocated on scratchpads
- → No polling
Distributed Message Passing

CPU_1
a=1
signal(sem_a)

CPU_2
wait(sem_a)
a=2
print a;

Shared Mem
a

sem_a

BUS
Distributed Message Passing (2)

CPU_{1} (prod)
- a = 1
- signal(sem_a)

CPU_{2} (cons)
- wait(sem_a)
- print a;

sem_a
- a = 1

BUS

- a = 1
You are given 2 implementations of the GSM codec
- Shared memory
- Distributed message passing
Simulate and compare these 2 approaches
- Energy
- Runtime
Application Mapping and Scheduling

\[
\begin{align*}
\tau_0 & \rightarrow \tau_1 \\
\tau_1 & \rightarrow \tau_2 \\
\tau_2 & \rightarrow \tau_3 \\
\tau_3 & \rightarrow \tau_4 \\
\tau_4 & \rightarrow \tau_5 \\
\tau_5 & \rightarrow \tau_0
\end{align*}
\]

\[
dl = 3 \\
dl = 6 \\
dl = 9
\]

CPU0: \(\tau_0, \tau_1, \tau_3, \tau_5\)

CPU1: \(\tau_2\)

CPU2: \(\tau_4, \tau_3\)
Mapping in MPARM Example

- Using the lightweight API of MPARM
  - `get_proc_id()`

```c
if (get_proc_id()==1) {
    //Task 1 executed on processor 1
    for (i=1;i<100;i++) a[i]=1;
}
if (get_proc_id()==2) {
    //Task 1 executed on processor 2
    for (i=1;i<100;i++) b[i]=1;
}
```
Thank you!
Questions?