Lab2 MPARM

Assignment 1: Design-Space Exploration for Energy Minimization

We started with a test of the default values, so that we could tweak caches sizes processor frequency and the cache mapping types. Running the simulation with no other flags set than –c1 -w would run the simulation using only one core, and logging power statistics. The default values were as follows:

Base test settings:

Frequency: **200 MHz**
Cache sizes: Data (*D*) **4096 Bytes**. Instruction (*I*) **8092 Bytes**.
Originally using: 4-way set associative (*D*), direct mapped (*I*).

Energy consumption:	
Core:	**16 264 066 [pJ]**
Data-cache:	**1 281 793 [pJ]**
Instruction-cache:	**23 564 524 [pJ]**
Sum:	**41 110 383 [pJ]**
Total (including RAM and other):	**45 038 855 [pJ]**

All tests are included in separate files, this document is mainly meant to summarise our findings, and the decisions that we chose to take.

Looking at the energy consumption we found out that the driving factors were Instruction cache and core. Data cache did not have as much of an impact, so we set out to prioritise bringing instruction cache and processor frequency down to save on power consumption.

We tried to minimize the energy consumption by making the instruction cache smaller.
When he instruction cache goes below 512 bytes, we see an increase in power consumption again. This might be caused by more missed reads forcing us to do many costly lookups instead of reading from the cache. We therefore settled on an instruction cache size of 512 bytes.

The data cache however, seems to not be possible to decrease below 512 bytes, and with little to no performance hit and some energy consumption minimisation, we determined to keep this as the maximum level.

4-way associative seems to be better to use on the data cache, this to be a good compromise between lookup time and the simplicity of direct mapping.

Direct mapping seems more suited for the instruction cache, maybe because the lookup is easier. Another reason might be that the instructions are not changed that much since we keep doing one task over and over.

Decreasing the clock speed bellow 200/11 = 18.1818... MHz seems to result in increased power consumption. This is probably caused by the fact that the processors power consumption does not scale linear with increased workloads.

Our optimal solution:

Frequency: **200/11** = **18.1818**... MHz
512 bytes data cache **4 way set associative mapping**.
512 bytes instruction cache **direct mapping**.
This setup resulted in an execution time of **17.93 milliseconds**, which was within the limits specified (**20 ms**).

Resulting in the following power consumption statistics:

Energy consumption:	
Core:	**4 627 895 [pJ]**
Data-cache:	**830 709 [pJ]**
Instruction-cache:	**7 972 474 [pJ]**
Sum:	**13 431 078 [pJ]**
Total (including RAM and other):	**17 738 707 [pJ]**

This results in an energy save of **45 038 855 - 17 738 707 = 27300148**

Assignment 2: Shared Memory vs. Distributed Message Passing

We choose to call Distributed Message Passing: DM and Shared Memory Based Communication: SM.

Energy Consumption

First we will consider the differences in the base case between DM and SM' energy consumption. SM seems to be more expensive to operate for the processor, costing more both when considering the energy consumption of the cores as well as the data transfer in the caches. SM is less expensive for the ram, since the semaphores are not stored in shared memory. However, this gain does not affect the overall power consumption to a degree that could outweigh the increased cost of operating the cores. For this base case DM is 289 µJ more expensive to use DM than SM.

Execution time

SM executes roughly 100 000 cycles/second more than DM. With SM executing 606306 cycles whereas DM executes 500869. Overall SM executes marginally less cycles than DM. However, DM executes ~4ms faster than SM, since SM does have to access the main memory more often with SM at ~10 ms and DM at ~14 ms. SM works "faster" but gets less work done since its being bottlenecked by the access speed of main memory.

Bus utilization

DM bus utilization 56.25%
SM bus utilization 44.80%
SM seems to use more of the bus. However, the bus utilization does not seem to be impacting performance to a sufficient degree that would make SM more efficient when running this process.

Improving SM traffic, synchronisation

Lowering the clock statically will decrease the bus utilization quite drastically but not really change the amount of semaphore reads that much, it seems that the read is done every time the processor wakes up a process. The process will then ask if it is time to start running again, and if not, it will go back to sleep.

We also tried to lower the clock frequency before issuing a wait and boosting it back up to normal when the process is signalled to start again. But we struggled to find a ratio where it would be beneficial to use this, it seems like it's more efficient to only lower the processor frequency statically. Since this seems to lower the amount of semaphore reads from 207333 to 186129. While keeping semaphore writes roughly the same.