Master Degree Program in Computer Science and Networking, 2016-17

High Performance Computing
Appello Special Session - October 31, 2016

Write your name, surname, student identification number (numero di matricola), e-mail. The answers can be written in English or in Italian. Please, present the work in a legible and readable form. All the answers must be properly and clearly explained, focusing on the requested issues.

1. Explain whether and why the following sentences are true or false:
   a) Two processing elements in a multiprocessor architecture execute atomic sequences of memory accesses to a shared data structure \( x \) contained in one C1 block. Process-to-processor mapping is based on exclusive mapping. “While one of the two PEs is within its atomic sequence, to provide atomicity the other PE is necessarily forced to wait on a busy-waiting condition”;
   b) In a multiprocessor architecture “the interrupt mechanism is used for receiving inter-processor communications and not for memory accesses generated by cache faults”. Explain whether and why the two operations are conceptually different.

2. A multi-CMP NUMA architecture has an external Generalized Fat Tree network based on a 2-ary n-fly butterfly. Explain:
   a) The number of input and output interfaces of a switch with double buffering communications;
   b) Which kinds of firmware messages the switch receives;
   c) Describe by words the very general idea of a routing algorithm executed by the switch unit. How the routing should be modified in a multi-CMP SMP architecture?

3. Let \( \Sigma \) be a system composed of four processes as depicted in the figure. Process \( P_1 \), with ideal service time \( T_{id-P_1} = 22M \tau \), generates a stream of integer elements transmitted to process \( P_2 \), which applies on each input element a computation with ideal service time \( 15M \tau \). The results are integer arrays of size \( M = 8K \) transmitted either to process \( P_3 \) with probability \( 2/3 \) or to \( P_4 \) otherwise. Process \( P_3 \) encapsulates and integer array \( B[M] \) statically initialized. The process applies the following computation on each input array \( A[M] \), where \( T_F = 400 \tau \):
   
   \[
   P_3:: \{ ... \text{int } A[M]; \text{ int } B[M];
   \]
   
   while true do
   receive(ch_in, A);
   for i=0 to M-1 do
   \( C[i] = F(A[i], B[M-i-1]); \)
   send(ch_out, C);
   
   \( P_4 \) has ideal service time \( T_{id-P_4} = 10M \tau \) and produces an output stream of arrays of size \( M \). Assume that \( P_1 \) and \( P_2 \) are allocated in the same CMP, while \( P_3 \) and \( P_4 \) are allocated in distinct CMPs.
   a) Study the system \( \Sigma \) by showing the ideal and effective service time and the relative efficiency of all the processes and of the system;
   b) Parallelize the process \( P_3 \) according to a data-parallel parallelization and provide the ideal/effective service time and the relative efficiency of the modules and of the system.

Architectural specifications
- Multi-CMP SMP multiprocessor architecture with 16 CMPs and external SMP network based on a binary butterfly with wormhole flow control (1 flit = 1 word);
- Each CMP has 8 PEs, an internal crossbar and 2 MINFs connected to 2 parallel WW units;
- Directory-based cache coherence with home flushing. Rdy-ack run-time support based on shared synchronization variables;
- Each PE has L1d of 32K and L2 of 1 Mega on-demand caches, \( T_{\text{inst}} = 2 \tau \), block size \( \sigma_1 = 8 \);
- External memory composed of 32 mutually interleaved macro-modules with clock cycle \( \tau_M = 20 \tau \);
- All the firmware interfaces use double buffering with \( T_{\text{tr}} \approx \tau \) inter-chip;
- Exclusive mapping;
- No communication processor.
Solution

1. The first sentence \( a \) is \textit{false}. Suppose that two processing elements \( PE_i \) and \( PE_j \) execute an atomic sequence of instructions like the one for manipulating a shared integer counter that must be atomically incremented:

\[
\text{LOAD RX, 0, Rval, set_indiv} \\
\text{INCR Rval} \\
\text{STORE RX, 0, Rval, reset_indiv}
\]

where \( set\_indiv \) and \( reset\_indiv \) are annotations indicating the beginning and the end of the atomic sequence. Suppose that \( PE_i \) enters the atomic sequence before the other processing element. The memory macro-module containing the cache block of the shared variable (in this example an integer \( x \)) receives the first memory request with indivisibility bit set to 1. Therefore, during the atomic sequence the memory macro-module serves only requests coming from \( PE_i \) eventually buffering all the requests coming from the other processing elements. In the meanwhile, \( PE_j \) is not executing any busy-waiting condition (e.g., a spin-loop test on a shared flag used for the synchronization), but the processing element waits for the end of the interpretation of the first LOAD instruction of its atomic sequence. In other words, in the LOAD interpretation the cache block transfer into C2/C1 of \( PE_j \) will be delayed until \( PE_i \) executes the STORE that resets the indivisibility bit.

Sentence \( b \) is \textit{true}. The interrupt mechanism is used to notify a processing element of an \textit{asynchronous} event that is not necessarily related to the code currently executed (i.e. of the currently running process). Consequently, the PE is interrupted by the event coming from its UC. The firmware phase of the interrupt handling mechanism is executed and the end of the interpretation of each assembler instruction, and tests the presence of an interrupt message. If the interrupt is present, the UC sends the message to the processor (IU) and the firmware interpreter of the current assembler instruction triggers the execution of the assembler handler (in the logical addressing space of all the processes) that will register the event received for future utilization (on multiprogrammed systems a process can be awaken and its PCB inserted in the ready list). Instead, a memory reading request-reply generated after a C2 cache fault is always a \textit{synchronous} event related to the interpretation of the currently executed assembler instruction (a LOAD) that generated the fault. Therefore, the block must be transferred from the memory to the C2 and C1 of the PE and the word transmitted to the processor.

2. The Generalized Fat Tree implementation is based on a binary butterfly, where at each level of the tree a switch node is composed in a modular fashion by a subset of the butterfly switches at the same level. A binary butterfly is composed of 2x2 elementary bidirectional switches (further decomposed in two independent units) and with double buffering interfaces. We have totally 16 level-transition firmware interfaces (8 inputs and 8 outputs) per switch. The switch is able to forward different kinds of messages:

- Remote memory reading request (3 flits) and reply (9 flits) messages for a C1 block from/to a C2 of a PE to/from a memory macro-module in the local memory of a CMP;
- Write-back request (11 flits) and reply (one flit) messages from/to the C2 of a PE to/from a macro-module in a remote memory;
- Write-through request (4 flits) and reply (one flit) messages from/to the C2 of a PE to/from a macro-module in a remote memory;
- Inter-processor communications (only requests without explicit reply, \( 1 + h \) flits with \( h \geq 1 \)) from a PE to another PE in a different CMP;
- Invalidation C2C request (3 flits) and reply (1 flit) from the C2 of a PE to the C2 of the home PE of the requested cache block;
- C2C read request (3 flits) and reply (9 flits) from the C2 of a PE to the C2 of the owner PE of the requested cache block;
- C2C store notification (11 flits) and reply (1 flit) from the C2 of a PE to the C2 of the home PE of the requested cache block.

In this architecture the external interconnection network is a Fat Tree and the routing is based on the \textit{common ancestor}. 


Once the first flit of a message arrives at a generic switch, the flit (header) is inspected and the destination identifier is compared with the index of the switch. We can distinguish between two possibilities:

i. the sub-tree rooted at that switch unit does not contain the destination node. In that case the message (flit-by-flit) is forwarded to the parent (actually, due to the modular design, the parent can be composed of multiple switches; the message will be forwarded to the correct switch according to the destination identifier);

ii. the sub-tree rooted at that switch unit contains the destination node. The flits of the message, received from the left (right) child switch are forwarded over the output double-buffering interfaces corresponding to the right (left) child switch.

Instead, in case of a multi-CMP SMP architecture, the external interconnection network is used in two ways:

i. according to the previous protocol (Fat Tree) in case of inter-processor communications generated by the local I/O of a PE and forwarded on the tree up to reaching the UC of the destination PE;

ii. according to the butterfly protocol, for example in case of a reading request/reply to/from a memory-macro module. In this case the routing is the standard one for butterfly networks (comparison among the bits of the source/destination identifiers to establish the correct link for routing, i.e. the straight or oblique link).

3. All the channels will be assumed with asynchrony degree 1. The inter-arrival time to process $P_2$ is equal to the ideal service time of $P_1$. $P_2$ is not a bottleneck:

$$\rho_2 = \frac{15M\tau}{22M\tau} = 0.68 < 1$$

The inter-departure time from $P_2$ is equal to its inter-arrival time. The inter-arrival time to process $P_3$ is given by the multiple-server theorem:

$$T_{A-3} = \frac{T_{D-2}}{P} = \frac{22M\tau}{\frac{2}{3}} = 33M\tau$$

Let us study in detail the behavior of $P_3$ by providing a D-RISC compilation of its source code:

NEW: <compilation of the receive RTS>

CLEAR Ri //Ri= index of the loop initialized to zero
SUB RM, 1, Rj //Rj= index for accessing B

LOOP: LOAD RA-addr, Ri, Ra //RA-addr contains the logical starting address of A
LOAD RB-addr, Rj, Rb, don’t_deallocate //RB-addr contains the logical starting address of B
CALL RF, Rret //RF contains the logical starting address of F’s code
STORE RC-addr, Ri, Ris //RC-addr contains the logical starting address of C
INCR Ri
DECR Rj
IF < Ri, RM LOOP
<compilation of the send RTS>
GOTO NEW

The function $F$ has been compiled as follows:

- the first input argument is a scalar value passed by using the register $Ra$;
- the second input argument is a scalar value passed by using the register $Rb$;
- the output result is a scalar value passed using the register $Rris$.

The pure calculation time is the following:

$$T_{calc-0} = M(7 \cdot T_{instr} + T_F) = M(14\tau + 400\tau) \approx 414M\tau$$
Per stream element we have *locality* in the blocks of $A$, *locality* in the blocks of $B$ (all the blocks are accessed exactly one time per stream element). Furthermore, we have *reuse* in the blocks of $B$ that are all accessed at every computation on a distinct stream element. Such reuse can be exploited both in C1 and in C2. The size of the input message is $M = 8K$ thus, assuming the channel between $P_2$ and $P_3$ is homed in the PE of $P_3$ its blocks can be found directly in C1 owing to home flushing. Therefore:

$$T_{fault} \approx 0$$

The calculation time is:

$$T_{calc} = T_{calc-0} + T_{fault} = 414M\tau$$

The communication latency cannot be overlapped and cannot be neglected. We suppose the PE of $P_3$ the home node of its output channel. Therefore, the stores in the target variable (write back) do not perform invalidation, i.e. $L_{com} = T_{send}(M) \approx M \cdot T_{transm} = 8M\tau$. In conclusion:

$$T_{id-P_3} = 422M\tau$$

The process is a bottleneck ($\rho_3 = 12.79$) and the optimal parallelism degree is:

$$n_{opt} = \left\lceil \frac{T_{id-P_3}}{T_{A-3}} \right\rceil = \left\lceil \frac{422M\tau}{33M\tau} \right\rceil = 13$$

To perform the steady-state analysis of the graph we correct the effective service time of the source:

$$T_{P_1} = T_{id-P_1,\rho_3} = 22M\tau \cdot 12.79 = 281M\tau$$

This is also the effective service time of $P_2$. The inter-arrival time to $P_4$ is:

$$T_{A-3} = \frac{T_{P_2}}{1 - p} = \frac{281M\tau}{\frac{1}{3}} = 844M\tau$$

And $P_4$ is not a bottleneck. The following table summarizes the steady-state analysis of $\Sigma$:

<table>
<thead>
<tr>
<th>Module</th>
<th>Ideal Ts</th>
<th>Effective Ts</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$22M\tau$</td>
<td>$281M\tau$</td>
<td>0.07</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$15M\tau$</td>
<td>$281M\tau$</td>
<td>0.05</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$422M\tau$</td>
<td>$422M\tau$</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>$10M\tau$</td>
<td>$844M\tau$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>$22M\tau$</td>
<td>$281M\tau$</td>
<td>0.07</td>
</tr>
</tbody>
</table>

The bottleneck can be parallelized according to a MAP data-parallel paradigm, with $M$ virtual processors where $V_P = \{A[i], B[M - i - 1]\}$ with $i = 0, ..., M - 1$. The array $B$ is statically divided into $n$ partitions each composed of $g = M/n$ elements. Each input array $A$ is scattered by a sequential scatter process. Suppose the following mapping:

- 1 CMP for $P_1$, $P_2$, $S$, $G$;
- 1 CMP with 8 workers;
- 1 CMP with 5 workers.

The home nodes are the following:

- CH-P1-P2 home node P2;
- CH-P2-S home node S;
- CH-S-Wi home node Wi;
- CH-Wi-G home node Wi;
- CH-G-OUT home node G;
- CH-P2-P4 home node P4.

The ideal service time of the scatter process is given by its communication latency. The blocks of the message can be found in its C1 (flushed by $P_2$). They must be written in the blocks of the target variable and flushed (synchronous flush stores) to the C1/C2 of the worker PE. So we have:

$$T_{id-S} = T_{send}(M) \approx \frac{M}{\sigma} \cdot L_{write-C2C}(\sigma)$$
The path has distance: C1-C2-W-INT_NET-MINF-WW-EXT_NETWORK-WW-MINF-W-C2-C1. We have \( d = 11 + d_{\text{net}} \). We have 8 paths with distance 3, and 5 paths with distance 5. Therefore, \( d_{\text{net}} \approx 4 \):

\[
L_{\text{write-C2C}}(\sigma) = (11 + 15 - 2) \cdot 2\tau + 20\tau + (1 + 15 - 2)2\tau = 96\tau
\]

The ideal service time of the scatter is \( T_{\text{id-sc}} = 12M\tau \) so it is not a bottleneck. A similar analysis can be performed for the gather process: in this case we pay \( M/\sigma \) C2C read requests from the C2 of the workers and we write the blocks locally (write back) because the gather is the home node for its output channel. Therefore, it is not a bottleneck.

The pure calculation time of a worker is:

\[
T_{\text{calc-w0}} = \frac{M}{n} (7 \cdot T_{\text{instr}} + T_F) = \frac{414M\tau}{n}
\]

The memory/cache hierarchy exploitation is the same of the sequential program. Each worker performs a communication of size \( M/(n \cdot \sigma) \) and the target variable is allocated in the worker PE. Therefore, such latency can be neglected, i.e. \( T_{\text{id-w}} \approx 414M\tau/13 = 31.8M\tau \) and this does not impact on the optimal parallelism degree.

<table>
<thead>
<tr>
<th>Module</th>
<th>Ideal Ts</th>
<th>Effective Ts</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>22M\tau</td>
<td>22M\tau</td>
<td>1</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>15M\tau</td>
<td>22M\tau</td>
<td>0.68</td>
</tr>
<tr>
<td>( P_3^{(13)} )</td>
<td>31.8M\tau</td>
<td>33M\tau</td>
<td>0.96</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>10M\tau</td>
<td>66M\tau</td>
<td>0.15</td>
</tr>
<tr>
<td>( \Sigma )</td>
<td>22M\tau</td>
<td>22M\tau</td>
<td>1</td>
</tr>
</tbody>
</table>