High Performance Computing

2nd midterm - December 21, 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. Two processes $P$ and $Q$ execute the following code working on a shared integer variable $y$ initialized to zero:

   ```java
   P:: y = 1; notify(event);
   Q:: wait(event); z = y + 1;
   ```

   Explain which memory ordering conditions are required to guarantee that the final value of $z$ is 2 and why they are needed in a multiprocessor architecture with automatic cache coherence (be precise not general).

2. A LC program $\Sigma$ is described by the figure below. Each process $\{P_1, P_2, P_3\}$ generates a stream of $m$ integer arrays $A[M]$ where $M = 128K$. The generic $P_i$ is defined as follows:

   ```java
   P[i]:: int $A[M]$ = <init>; channel out chIN[i];
   while true do \{ $\forall$ $i = 0 ... M - 1$: $A[i] = F_i(A[i])$; send(chIN[i], A); \}
   ```

   Any function $F_i(i = 1, 2, 3)$ executes $40$ assembler instructions on average. Process $Q$ executes the following code where $G$ has average calculation time $T_G = 200r$ and $B[M]$ is an initialized internal array:

   ```java
   Q:: int $A[M]$; $B[M]$ = <init>, $C[M]$; channel in chIN[3](1); channel out chOUT;
   while true do \{ alternative $i = 0, 1, 2$ receive(chIN[i], A) do skip;
   \forall i = 0 ... M - 1$: $C[i] = G(A[i], B[M - i - 1])$; send(chOUT, C); \}
   ```

   Received an array $C$ from $Q$, the process $R$ computes $y = \sum_{i=0}^{M-1} C[i]$ and writes $y$ in the next entry of a private array $Z[3m]$. $\Sigma$ is executed on the multi-CMP machine specified below by mapping $\{P_1, P_2, P_3\}$ on the same CMP while $Q$ and $R$ are executed on different CMPs. The PE of $Q$ is the home node of all the communication channels.

   a) Study the system behavior in terms of the base latency. Evaluate the effective service time and relative efficiency of the five processes and of the whole system;

   b) Determine the parameters $p_T, R_{QD}, T_S$ of the under-load latency cost model and identify the most stressed cache.

3. A process $P$ receives a stream of arrays $A$ of $M$ integers and encapsulates an array $B$ of the same size statically initialized. For each array $A$, process $P$ executes the following computation where $B$ is transmitted onto the output channel of the process:

   ```plaintext
   for $i$ = 0 to $M-1$
   for $h$ = 0 to $M-1$
   $B[i] = f(A[h], B[i])$;
   ```

   a) Describe a map Virtual Processor version in a LC-like formalism;

   b) Describe a stencil-based Virtual Processor version in a LC-like formalism and explain which kind of stencil you identify in this program.

Architectural specifications for question 2

- Multi-CMP NUMA-SMP machine with 16 CMPs and an external NUMA network based on a Generalized Fat Tree with double-buffering firmware interfaces, 1-word flit wormhole flow control;
- Each CMP has 4 PEs, an internal crossbar and 2 MINFs connected to 2 off-chip WW units;
- Each PE has L1d of 32K and L2 of 1Mega on-demand caches, $T_{\text{instr}} = 2r$, block size $\sigma_1 = 8$;
- 64-Gigaword local memory for each CMP interconnected with the two WW units by a crossbar;
- Directory-based automatic cache coherence with home-flushing semantics;
- Exclusive mapping, RDY-ACK run-time support based on shared memory synchronization;
- Each communication processor shares the L1/L2 caches with the “main” processor of the PE.
Solution

1. We assume that the notify and wait primitives operate on a shared boolean flag (EV) implementing the event. It is necessary that the store instruction that sets EV to true (STORE2) in the notify is made visible to the PE of Q after the store instruction that writes the value 1 into the variable y (STORE1).

Suppose that the variable y is allocated in the block $b_y$ and the event in the block $b_{ev}$. Without loss of generality the two blocks are homed in two distinct PEs, i.e., let say PE_Y and PE_EV. Although STORE1 starts before STORE2, its firmware interpreter can be completed after the one of STORE2. In fact, a possible scenario is the following: STORE1 generates a store notification to PE_Y that can be received (e.g., due to the non-deterministic behavior of the interconnection network) after the store notification generated by the interpreter of STORE2 to PE_EV. In that case, the process Q, which is running a busy-waiting loop on the EV flag, may receive the invalidation and may load again the block of EV before the store notification on y arrives at PE_Y. Therefore, the process Q can terminate the wait and load under the control of PE_Y the old value of $b_y$ which still contains the value zero. Under these conditions, the result written in z is 1 and not 2. To avoid this behavior STORE1 must be synchronous, i.e., the interpreter of STORE2 can start only after the completion of the interpretation of STORE1, i.e., when the modification of y is updated in the directory in the home node PE_Y and its visible to all the PEs.

2. In the analysis we will assume that all the channels have asynchrony degree $k = 1$. All the processes $P_i$ have the same behavior:

START: <compute A>;
   <send A onto chIN[i]>;
GOTO START

The compilation of the compute phase is described as follows:
& $\sigma$-unfolding, write_back asynchronous & {
   CLEAR Ri
   LOOP: LOAD RA-addr, Ri, Ra
   CALL RFi, Rret
   STORE RA-addr, Ri, Rout, don’t_deallocate
   INCR Ri
   IF < Ri, RM LOOP
}

The function $F_i$ is compiled in order to receive the input argument from the register Ra while the output result is written in the register Rout. The pure calculation time consists in five instructions of the loop body:

$$T_{calc-Pi-0} = M(5 \cdot T_{instr} + T_F) = M(10\tau + 80\tau) = 90M\tau$$

The computation is characterized by reuse in the blocks of A, which is read completely (all the blocks) at each service time. Reuse can be exploited in C2 but not in C1 for capacity reasons, i.e., we pay the faults from C2 to C1 during the compute phase:

$$T_{calc-Pi} = T_{calc-Pi-0} + \frac{M}{\sigma}L_{C2-C1}(\sigma) = 90M\tau + 1.25M\tau = 91.25M\tau$$

The calculation part must be compared with the communication latency. The architecture allows communication overlapping, owing to the presence of a communication processor sharing C1/C2 with the “main” processor of each PE. The don’t_deallocate flag makes it possible to be sure (with high probability) that all the blocks of A are still in C2 at the end of the compute phase. Therefore:

$$L_{com-Pi} = T_{send}(M) \approx M \cdot T_{trans} = \frac{M}{\sigma}(L_{C2-C1}(\sigma) + L_{write-C2-C2}(\sigma)) + M \cdot \beta_{code}$$

All the blocks of A are read in C1 by the communication processor and flushed with self-invalidation to the C2 of process Q which is the home node. The PATH of a flush operation is: C1-C2-W-INT_NET-MINF-WW-EXT_NUMA_NET-WW-MINF-INT_NET-W-C2, i.e., $d = 11 + d_{numa}$. The distance in the
NUMA network (fat tree) depends on the mapping, and the average path with the uniform distribution $1.9 \cdot n = 10$ is too pessimistic (with $n = 5$ in this case). A possible mapping is to choose $CMP_0$ for mapping processes $P_1, P_2, P_3$ while $Q$ and $R$ are mapped onto $CMP_1$ and $CMP_2$ respectively. The distance in the NUMA network is $d_{numa} = 3$. Therefore, we have:

$$L_{\text{write-C2-C2}}(\sigma) = (11 + d - 2)T_{hop} + (1 + d - 2)T_{hop} = 72\tau$$

And the communication latency is:

$$L_{\text{com-Pi}} = 18.25M\tau$$

which can be completely masked with the internal calculation. Thus:

$$T_{\text{id-Pi}} = 91.25M\tau$$

The inter-arrival time to process $Q$ is obtained by using the multiple-client theorem of acyclic graphs (OR semantics):

$$T_{A-Q} = \frac{1}{3} = \frac{T_{\text{id-Pi}}}{3} = \frac{91.25M\tau}{3} \approx 30.5M\tau$$

Process $Q$ has the following behavior:

START: <receive A from any $P_i$>

<compute C>;

<send C onto chOUT>;

GOTO START

The compilation of the compute phase is described as follows:

& $\sigma$-unfolding, write_back asynchronous & {

CLEAR Ri

SUB RM, 1, RM-1

LOOP: LOAD RA-addr, Ri, Ra

SUB RM-1, Ri, Rj

LOAD RB-addr, Rj, Rb, don’t_deallocate

CALL RG, Rret

STORE RC-addr, Ri, Rout, don’t_deallocate

INCR Ri

IF < Ri, RM LOOP

}

Register Ri is used for indexing the loop, while register Rj is used to access the elements of the array $B$. The function $G$ takes the two input arguments from the registers Ra and Rb and produces the output result in the register Rout. The pure calculation time is:

$$T_{\text{calc-Q-0}} = M(7 \cdot T_{\text{instr}} + T_G) = M(14\tau + 200\tau) = 214M\tau$$

The computation is characterized by locality on $A$, where the blocks of the array are accessed from the first one to the last one. Similarly, per service time we access all the blocks of $B$ (from the block of the last element to the first block). The blocks of $B$ can be likely contained in C2 but not in C1. Therefore, we have reuse on the blocks of $B$ exploitable in C2 only. By taking into account the fault overhead we have:

$$T_{\text{calc-Q}} = T_{\text{calc-Q-0}} + 2\frac{M}{\sigma}L_{C2-C1}(\sigma) = 214M\tau + 2.5M\tau = 216.5M\tau$$

Note that we pay $2M/\sigma$ faults for transferring from C2 to C1 the blocks of $A$ and $B$. The blocks of $A$ have been flushed in the C2 cache (home node) by the sending process $P_i$.

The communication latency of the array $C$ is given by:

$$L_{\text{com-Q}} = T_{\text{send}}(M) = M \cdot T_{\text{trasn}} = \frac{M}{\sigma} \left(2 \cdot L_{C2-C1}(\sigma)\right) + M \cdot \beta_{\text{code}}$$

Observe that in the send execution we pay the C2-C1 transfers of the cache blocks of the array $C$ during the message reading. Furthermore, since the C2 of process $Q$ is the home node of the blocks of the target
variable of channel chOUT, we pay the latency of $M/\sigma$ synchronous write-back C1-C2 store instructions. The communication latency is:

$$L_{com-Q} = 10.5 M\tau$$

That obviously can be overlapped with the internal calculation:

$$T_{id-Q} = 216.5 M\tau$$

Process R has the following behavior:

START: <receive C from any Q>  
   <compute y and store it in Z>;  
   GOTO START

The compilation is the following:

CLEAR Rc  
NEW: <receive array C>  
   CLEAR Ri  
   CLEAR Ry  
LOOP: LOAD RC-addr, Ri, Rc  
   ADD Ry, Rc, Ry  
   INCR Ri  
   IF < Ri, RM LOOP  
   STORE RZ, Rk, Ry  
   INCR Rk  
   GOTO NEW

Register Rk is used to index the array $Z$, register Ry contains the variable y initialized to zero. Register Ri is used as the index of the loop. During the loop, we pay $M/\sigma$ C2C read requests from C1 of PE_R to C2 of PE_Q which is the home node of the vtg of chOUT. The distance between $CMP_1$ and $CMP_2$ is $d_{numa} = 5$. The latency of a C2C read request is:

$$L_{C2C-1}(\sigma) = (3 + d - 2)T_{hop} + (9 + d - 2)T_{hop} = 80\tau$$

And we obtain:

$$T_{id-R} = T_{calc-R} = M (4 \cdot T_{instr}) + \frac{M}{\sigma} \cdot L_{C2C-1}(\sigma) = 18 M\tau$$

The analysis with base latency can be summarized as follows:

<table>
<thead>
<tr>
<th>Module</th>
<th>Ideal Ts</th>
<th>Effective Ts</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1, P_2, P_3$</td>
<td>$91.25 M\tau$</td>
<td>$650 M\tau$</td>
<td>0.14</td>
</tr>
<tr>
<td>$Q$</td>
<td>$216.5 M\tau$</td>
<td>$216.5 M\tau$</td>
<td>1</td>
</tr>
<tr>
<td>$R$</td>
<td>$18 M\tau$</td>
<td>$216.5 M\tau$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>$30.5 M\tau$</td>
<td>$216.5 M\tau$</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Process Q is obviously a bottleneck. The three clients (identical) at steady-state must have an effective service time $T_{p_1}$ such that:

$$T'_{A-3} = \frac{T_{p_1}}{3} = T_{id-Q} = 216.5 M\tau$$

Therefore, $T_{p_1} = 650 M\tau$.

For the under-load analysis we have to find the parameters of the cost model. The following figure shows the interactions among C2s and for the most stressed secondary cache the interactions with its C1. It is straightforward to discover that the C2 of process Q is the most stressed cache.
We identify three different types of cache-to-cache interactions:

- Synchronous non-local stores with home-flush annotation: \( L_1 = L_{\text{write}C2C-C2}(\sigma) = 72\tau \)
- Synchronous C1-C2 write-back local stores and C2-C1 read requests: \( L_2 = L_{C2-C1}(\sigma) = 10\tau \)
- C2C read requests: \( L_3 = L_{C2C}(\sigma) = 80\tau \)

The number of interactions of the various types is:

\[
\begin{align*}
\rho_1 &= \frac{3M}{\sigma} \\
\rho_2 &= \frac{5M}{\sigma} \\
\rho_3 &= \frac{M}{\sigma}
\end{align*}
\]

The frequencies are: \( \pi_1 = 1/3, \pi_2 = 5/9, \pi_3 = 1/9 \) and the average base latency is:

\[
R_{Q-0} = \sum_{i=1}^{3} \pi_i \cdot L_i = 38.44\tau
\]

To derive \( T_p \) we measure the requested bandwidth by each client using the corresponding effective service times described in the previous table:

\[
\begin{align*}
B_{\text{req}P_i} &= \frac{C_{P_i}}{T_{P_i}} = \frac{M}{650M\tau} = \frac{1}{650\tau \cdot \sigma} \\
B_{\text{req}Q} &= \frac{C_{Q}}{T_{Q}} = \frac{5M}{216.5M\tau} = \frac{5}{216.5\tau \cdot \sigma} \\
B_{\text{req}R} &= \frac{C_{R}}{T_{R}} = \frac{M}{216.5M\tau} = \frac{1}{216.5\tau \cdot \sigma}
\end{align*}
\]

The average time between two consecutive requests arriving at the secondary cache of process Q is:

\[
T_p = \frac{1}{B_{\text{req}tot}} = \frac{1}{B_{\text{req}P_i} + B_{\text{req}Q} + B_{\text{req}R}} = 273\tau
\]

The other parameters of the cost model are \( p = 5 \) and \( T_S = 8\tau \). The solution of the system of equations (optional, according to page 367 of the textbook for the M/D/1 model) gives the following results:

- \( \alpha = R_Q/R_{Q-0} \approx 1.02 \)

Therefore, caches introduce a negligible contention and the analysis with the base latency is accurate of the real execution.

Furthermore, it is easy to observe that macro-modules do not introduce any additional congestion in this application. Processes \( P_i \) loads the blocks of the initial array A only for their first stream element, and thereafter those blocks remain in their C2 for the whole execution. Process \( R \) writes a block of the array Z in memory asynchronously every \( \sigma \) arrays C from Q (thus such write-backs are very unfrequent). The C2 cache of process Q, which is the home node of all the channels, updates the blocks of A and of C in the local memory of its CMP in parallel with the services requested by the other caches. Check as an exercise that such asynchronous interactions with the memory do not generate an overhead for C2-Q, i.e they can be overlapped with the execution.

3. A first data parallel version is characterized by \( M \) virtual processors where the i-th virtual processor has the i-th element of array \( B \) (read and write, owner computer rule respected), and all the elements of the
array $A$. The computation is a \emph{map}, i.e. $VP_i$ at each computation step (the iterations of the inner-most loop) reads a different element of the array $A$ which is owned by the VP itself. The pseudo-code of a generic VP according to a LC-like syntax is:

```c
parallel VP[M]; int A[M], B[M]=<initialized>; channel input_stream; channel output_stream;
VP[i][i=0...M-1]: int A[i], B[i]; channel in input_stream(1); channel out output_stream;

while true do {
    receive(input_stream, A);
    for h=0 to M-1 do {
        B[i] = F(A[h], B[i]);
    }
    send(output_stream, B[i]);
}
```

The computation time is reduced from $O(M^2)$ to $O(M)$.

Another data parallel version is characterized by $M$ virtual processors where the i-th virtual processor has the i-th element of array $B$ (read and write, owner computer rule respected), and the i-th element of the array $A$. The computation is a \emph{stencil}, i.e. $VP_i$ at each computation step (the iterations of the inner-most loop) reads a different element of the array $A$ owned by a different VP. Except for the step $i = h$, where $VP_i$ accesses its local element, a communication is needed to read the required element of the input array. The pseudo-code of a generic VP according to a LC-like syntax is:

```c
parallel VP[M]; int A[M], B[M]=<initialized>, x; channel input_stream; channel output_stream; channel stencil[M][M];
VP[i][i=0...M-1]: int A[i], B[i]; channel in input_stream(1); channel in stencil[*][i](1); channel out output_stream; channel out stencil[i][*];

while true do {
    receive(input_stream, A);
    for h=0 to M-1 do {
        if (h == i) then {
            multicast(stencil[i][*], A[i]);
            B[i] = F(A[i], B[i]);
        } else {
            receive(stencil[h][i], x);
            B[i] = F(x, B[i]);
        }
    }
    send(output_stream, B[i]);
}
```

Also in this case the computation time is reduced from $O(M^2)$ to $O(M)$. 