Master Degree Program in Computer Science and Networking, 2014-15

High Performance Computing

4th appello – July 9, 2015

Write your name, surname, student identification number, 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.

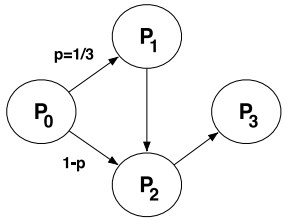
1) A multiprocessor system is defined according to the Architectural Specifications below and according to the following additional constraint on the parallel programs: *any shared data structure is shared, in read/write mode, by two processes only.*

a) Describe the firmware behavior of a generic secondary cache unit C2. The description *i)* must be at a sufficient level of detail to understand the *main hardware resources* for cache control and the clock cycles spent for the various operations, *ii)* must take into account *private and shared data*, and *iii)* must be given for *home* nodes and for *non-home* nodes of shared data.

b) Evaluate the service time of Load and Store instructions executed by a *home* node, both for *private* and for *shared* data, in all the possible situations of data allocation in the memory hierarchy, assuming $R_Q = R_{Q0}$;

c) Evaluate and justify the maximum bandwidth of C2, i.e. number of started operations per clock cycle.

2) The system Σ, at process level, is composed of four sequential modules: P_o, P_1, P_2 and P_3, interconnected as shown in the figure. Module P_o generates an infinite stream. The generic *i*-th stream element is composed of a pair of arrays $A_i[M], B_i[M]$ with M = 512 integers. P_o transmits each stream element to P_1 with probability $p = 1/3$ or to P_2 otherwise. P_2 receives non-deterministically from P_0 or P_1. P_o and P_1 have ideal service times equal to $1M^2\tau$ and $6M^2\tau$ respectively and *they cannot be parallelized*. All streams have asynchrony degree equal to one. P_3 has a negligible ideal service time. Module P_2 executes the following computation:




```
P2 :: while(true) {
        receive (A,B) from either Po or P1;
        for i=0 to M-1 do
              for j=0 to M-1 do
                    C[i,j]=F(A[i],B[j]);
        send (C) to P3;
        }
```

The mean calculation time of function *F* is 10τ. The Architectual Specifications are the same of Question 1) except that point 5 is modified as follows: directory-based cache coherence with *home-flushing semantics*.

a) Determine the optimal parallelism degree of P_2.

b) Discuss *all* the feasible parallel implementations of P_2, and study in detail a *data parallel* implementation. This issue must be studied carefully, including (but not limited to) the impact of the memory hierarchy and of the communications.

c) According to the base latency, evaluate the effective service time and the relative efficiency of P_o, P_1, P_2 (parallel), P_3 and of the whole system Σ.

Architectural Specifications

1. Single-CMP with 16 PEs, 4 MINFs, bi-directional ring wormhole interconnect with double buffering. On the ring, four PEs alternate with one MINF;
2. SMP architecture with 1Tera-word external memory, 4 shared interleaved macro-modules, 8 modules each and clock cycle of 10τ;
3. each PE is D-RISC with on-demand inclusive 2-level cache, 32K+32K C1, 512K C2, 8-word C1-blocks. Each PE is equipped with a dedicated communication processor;
4. the mean service time per instruction, in absence of cache faults, is equal to 2τ;
5. directory-based cache coherence with basic invalidation semantics. The cache coherence control is *entirely* delegated to, and only to, the secondary cache units;
6. lock-free synchronization;
7. exclusive mapping.

Solution

1) Since the global architecture is SMP, the four shared external memory macro-modules are interleaved each other (not only inside a single macro-module). Any external reference by a PE can refer any macro-module with the same probability (1/4).

The *directory-based* scheme is implemented by associating any PE a partition of the physical address space *uniformly*, i.e. 1Tera/16 = 64G words (8G blocks) per partition.

According to the specifications, the distribution of PEs and MINFs on the bi-directional ring is the following:

… W0, W1, W2, W3, MINF0, W4, W5, W6, W7, MINF1, W8, W9, W10, W11, MINF2, W12, W13, W14, W15, MINF3, W0, …

The average distance of a PE from a MINF is 10/4 with probability 1/2 (e.g. group W0, W1, W2, W3 wrt MINF0 or MINF3), and 30/4 with probability 1/2 (e.g. group W0, W1, W2, W3 wrt MINF1 or MINF2). Thus the average distance of a PE from a MINF is

$$d_{internal\ \_net} = 5$$

Since the synchronization is lock-free, no firmware support to indivisible sequences of memory accesses exists (e.g. no *indiv* bit is associated to the memory references).

The additional constraint imposed on the parallel programs is summarized as follows: *a PEi and a PEj share a data structure* (a set of data structures, including synchronization if implemented via shared variables) *Sij in read/write mode, and no PEk, with $k \neq i$ and $k \neq j$, shares Sij*.

Let us assume that *PEi* is the *home node* of *Sij*.

a,b) We have to describe the behavior of C2i (home controller of Sij) and of C2j (non home): precisely, *the part of Load and Store firmware interpreter which is executed by the secondary cache unit*. Accordingly, we have to evaluate the service time of Load and Store for PEi.

C2 Operation Part contains all the hardware resources to implement the cache controls, including cache coherence, *in a single clock cycle*: notably, an *associative memory* for the directory partition. The associative access to this table is done using the block identifier, which is the most significant part (37 bits) of the physical address; each entry contains the cache management and cache coherence state information (shared, modified, allocation node (s)).

C2 is connected, in input and in output, to C1 and to W. The behavior is *nondeterministic*, with the possibility to serve requests from C1 and from W *in parallel* in the same clock cycle, according to the type of operation. Even when C2 is waiting for an answer from the other C2, it is possible that another request is received from C1 and served in parallel.

We will use the abbreviations

Load b *Store b*

where *b* is a block identifier.

Private block (let C2 = C2i or C2j: each node is home of its own private data)

C2 operation requests, from C1, are generated by *Load* and *Store* instructions, executed by the PE processor, which caused block fault in C1.

Load b executed by PE

- no fault in C1: C2 is not involved. The instruction service time *T* is not penalized, thus it is equal to 2τ (see specifications);
- fault in C1 and no fault in C2: C2 receives the block transfer request from C1 and, in the same clock cycle, starts transferring the block value to C1, thus

$$T = 2\tau + (\sigma_1 + 2)\,\tau = 12\,\tau$$

- fault in C1 and in C2: C2 receives the block transfer request from C1, in the same clock cycle starts transferring the block from the external memory M (through W and the appropriate MINF) and copies the block stream to C2 itself and, in parallel, to C1 word by word.

We have to evaluate the latency of a C1-block transfer from M. The path of a block read request is C1, C2, W, internal net, MINF, IM, M, thus (see analysis of ring distance) with distance d = 11. Therefore:

$$T = 2\tau + L_{extM-C1}(\sigma_1) = 2\tau + \left(s_{req} + s_{reply} + 2\,d - 4\right) T_{hop} + \tau_M = 72\,\tau$$

since s_{req} = 3 (header, physical address), s_{reply} = 9 (header, block words), T_{hop} = 2τ (maximum hop along the path).

Store b executed by PE

- fault or no fault in C1: direct execution (and block allocation in case of fault) in C1 and in C2 in parallel. In the same clock cycle C2 serves a single-word (write-through) or starts a C1-block (write-back) write request, respectively:

$$T\,(1) = 2\tau + \tau = 3\tau \qquad T(\sigma_1) = 2\tau + \sigma_1\,\tau = 10\,\tau$$

The main memory is updated asynchronously. There is no impact on the instruction service time provided that the under-load bandwidth of M is greater than the external Store rate.

Shared block: C2i home

Again, C2i operation requests, from C1, are generated by *Load* and *Store* instructions executed by the PEi processor and which caused block fault in C1.

Moreover C2i receives requests from C2j, through Wi, for *C2C block reading* (PEj Load) and for *Store notification* (PEj Store).

Load b executed by PEi

- no fault in C1: identical to private data case - *b* is updated, thus C2i is not involved (T = 2τ);
- fault in C1 and no fault in C2: identical to private data case - *b* is updated in C2i, which receives the block transfer request from C1 and transfers the *b* value to C1 (T = 12τ);
- fault in C1 and in C2: C2i receives the block transfer request from C1 and, in the same clock cycle, verifies whether C2j is the owner of *b*: in this case, in the same clock cycle C2i requests a C2C block transfer to C2j and waits the block stream word by word, copied to C2i and to C1:

$$T = 2\tau + L_{C2C-C1}(\sigma_1) = 2\tau + \left(s_{req} + s_{reply} + 2\,d - 4\right) T_{hop} = 30\tau$$

where d = *10* (path C1, C2, W, internal net, W, C2) and T_{hop} = τ.

Otherwise (*b* is not in C2j) C2i behaves as with private data, i.e. block transfer from M:

$$T = 72\,\tau$$

Store b executed by PEi

- fault or no fault in C1: direct execution (and block allocation in case if fault) in C1 and in C2i, which becomes/remains the owner of *b*. In the same clock cycle C2i serves a single-word (write-through) or starts a C1-block (write-back) write request. In parallel, C2i verifies whether *b* is currently allocated in C2j: if it is not, we have the same service time for private data.

  If *b* is currently in C2j, in the same clock cycle C2i sends and invalidation message to C2j (through Wi), waits the ack and then sends the ack to C1.

  The write operation and the synchronous invalidation protocol occur in parallel: the service time is equal to the maximum between service time for private data and the latency for synchronous invalidation protocol. We have:

$$L_{invalidation} = L_{inv-req} + L_{inv-ack}$$

The request length is 3 flits, the ack just 1 flit. The path is C2, W, internal net, W, C2, with distance $d = 9$. Thus:

$$L_{invalidation} = (s_{req} + s_{reply} + 2\,d - 4)\,T_{hop} = 18\tau$$

and, for write-through or for write-back:

$$T = 18\tau$$

C2C block transfer generated by a PEj Load

- C2i receives from C2j a block transfer request. In the same clock cycle it starts sending the block to W word by word by C2C, if *b* is currently allocated in C2i, or by transfer from M, and updates the local knowledge.

Store notification from C2j

- C2i receives from C2j a Store notification. In the same clock cycle C2i updates the local knowledge of *b* state (C2j ownership), and, if *b* is currently allocated in Pei, invalidates *b* in C2i and in C1.

Shared block: C2j non-home

Load b executed by PEj

- no fault in C1: identical to private data case - *b* is updated, thus C2j is not involved;
- fault in C1 and no fault in C2j: identical to private data case - *b* is updated in C2j, which receives the block transfer request from C1 and transfers the block value to C1;
- fault in C1 and in C2: C2j receives the block *b* transfer request from C1, and in the same clock cycle it forwards the requests to C2i (home), then it waits the block word by word;

Store b executed by PEj

- fault or no fault in C1: direct execution (and block allocation in case if fault) in C1 and in C2j, which becomes/remains the owner of *b*. In the same clock cycle C2j serves a single-word (write-through) or starts a C1-block (write-back) write request. In parallel, C2j send a Store notification to C2i (home);

C2C block transfer generated by a C2i (home) Load

- C2j receives through Wj the C2C block request from C2i and, in the same clock cycle, starts replying with the block stream word by word.

Invalidation from C2i (home)

- C2j receives through Wj the invalidation of *b*, which will be deallocated in C2j and in C1, and, in the same clock cycle, sends the invalidation ack to C2i.

c) As said, C2 is able to start, in the same clock cycle, an operation requested by C1 on private/shared data and an operation requested/replied by the partner C2 for shared data. Thus, the maximum bandwidth is

2 started operations/clock cycle

This transformation of nondeterminism into parallelism can be done in the same C2 unit at the clock cycle level. If two units are used, they maintain updated a replicated part of the directory table.

2) As a first step we need to determine if P_2 is a bottleneck or not.

Inter-arrival time to P_2

We use the results of the graph analysis explained in Sections 4.2 and 4.3 of the book. The inter-arrival time to P_1 is given by:

$$T_{A-1} = \frac{T_{P_0}}{p} = \frac{1M^2\tau}{\frac{1}{3}} = 3M^2\tau$$

The utilization factor of P_1 is:

$$\rho_1 = \frac{T_{P_1}}{T_{A-1}} = \frac{6M^2\tau}{3M^2\tau} > 1$$

Thus P_1 *is a bottleneck* and we need to correct the service time of the source:

$$T_{P_0}^{'} = pT_{P_1} = \frac{1}{3}6M^2\tau = 2M^2\tau$$

Finally, the inter-arrival time to P_2 is given by:

$$T_{A-2} = \frac{1}{\frac{1-p}{T_{P_0}^{'}} + \frac{1}{T_{P_1}}} = \frac{1}{\frac{2/3}{2M^2\tau} + \frac{1}{6M^2\tau}} = 2M^2\tau$$

Ideal service time of P_2

Let P_2 be the home node of the three shared *structs* VTG_in1, VTG_in2 and VTG_out, respectively for the two input streams and the output stream, each one containing one elementary VTG_S structure (unitary asynchrony degree for each channel).

In any zero-copy implementation of the run-time support(including Rdy-Ack), the *receive* set_ack phase can be neglected with respect to the compute phase, which has a completion time in the order of $O(M^2)$. Analogously, the *send* setup phase has a negligible impact compared to the copy phase.

P_2 is compiled as follows:

```
&σ-unfolding, write_back &
{
                LOAD   R_vtg, 0, RA
                LOAD   R_vtg, RM, RB
                CLEAR   Ri
  LOOP_i:       CLEAR   Rj
                LOAD  RA, Ri, Ra          //RA contains the initial logical address of array A.
LOOP_j:         LOAD  RB, Rj, Rb          //RB contains the initial logical address of arrayB.
                CALL RF, Rret             /*RF contains the initial logical address of function F (input parameters
                STORE   RC, Rj, Rc        in the registers Ra, Rb and output result in Rc).*/
                INCR   Rj
                IF <   Rj, RM, LOOP_j
                ADD RC, RM, RC            //RC points to the next row of the matrix C (row-major representation).
                INCR   Ri
                IF <   Ri, RM, LOOP_i
}
```

Matrix C is represented according to the "Row Major" order, i.e. all the integers on the same row are stored contiguously in the virtual memory of the process.

By focusing on the innermost loop the pure code has service time:

$$T_{calc\,-0} = M^2(T_F + 10\tau) = 20M^2\tau$$

The working set consists in a block of A, *all the blocks of B* and one block of C. We exploit spatial locality in A and C and reuse of B, which can be contained both in C2 and in C1. During the whole execution the process P_2 generates M/σ cache faults during the load instructions of A in the outermost loop ($\sigma = \sigma_1$). Other M/σ cache faults of B blocks are paid during the fist iteration of the outermost loop.

For the two input streams the target variables are can be contained both in C2 and in C1. According to the home flushing semantics, the blocks of A and B have been flushed by P_o or P_1 directly into the C2 and C1 of P_2 during the *send* primitive and deallocated from their local caches. Therefore, no penalty for C1 faults is paid, and :

$$T_{calc} \sim T_{calc\,-0} = 20M^2\tau$$

In the send phase to P_3 the setup phase of the communication can be neglected. We consider the copy of the matrix C that fits in C2 but not in C1. The copy includes one write-back store into C2 without cache coherence overhead every σ iteration of the inner-most loop:

$$T_{copy}\,(M^2) = M^2 T_{trasm} \sim M^2(\beta_{code})$$

No invalidation is needed, since the destination process provides to invalidate the received value as soon as it is copied into the result variable. Therefore, the copy latency is

$$T_{copy}\,(M^2) \sim 10M^2\tau$$

In conclusion we have:

- receive latency has negligible impact;
- calculation time is $T_{calc} = 20M^2\tau$;
- send latency reduces to the message copy only: $T_{copy}\,(M^2) = 10M^2\tau$

With communication processor we have:

$$T_{P_2-id} \sim max\{T_{calc}, T_{copy}\,(M^2)\} = 20M^2\tau$$

a) Optimal parallelism degree of P_2

The optimal parallelism degree of P_2 is given by:

$$n_{opt} = \left\lceil \frac{T_{p2-id}}{T_{A-2}} \right\rceil = \left\lceil \frac{20M^2\tau}{2M^2\tau} \right\rceil = 10$$

The number of PEs ($N = 16$) is sufficient to accommodate the optimal parallelism degree value (only three PEs are needed by the other processes of Σ).

b) Parallel implementation of P_2

P_2 can be parallelized according to different paradigms:

1. *farm*: the computation on each pair A,B is *stateless*. It is easy to show that the emitter will not be a bottleneck, as it performs a send with a copy in the order of $O(M)$ while the workers executes a code which is $O(M^2)$. Therefore the optimal parallelism degree can be exploited and the bottleneck removed. Also the send to the collector can be overlapped. This solution has latency roughly equal to the one of the sequential implementation. Two service processes are needed (emitter and collector) and they can be accommodated in the architecture (3+2+10=15<N=16);
2. *pipeline with loop unfolding of the outermost loop:* no service processes are needed. This implementation can be difficult or not practical. Each stage should receive the arrays A and B and produces a "partially filled" C. Each stage applies the computation to produce a partition (set of contiguous rows of C). The reuse of B has been already exploited in the sequential version, so no particular memory hierarchy implication results from the data partitioning. With $n_{opt} = 10$ the ideal

service time of each worker is roughly equal to $2M^2\tau$ plus the transmission of the message (A,B and in the non-optimized solution the whole C) which is greater and cannot be fully overlapped. Therefore, without particular optimizations this solution does not exploit the optimal degree of parallelism. Moreover, its latency is $O(n)$, higher than the sequential implementation;

3. *data parallel map:*a scatter process distributed the arrays A and B to the workers, and a gather process collects the partitions of C. This solution exploits the ideal parallelism degree if the input/output distribution is not a bottleneck, which is quite obvious since the copy of A and B is one order of magnitude lower than the computation in the workers. The reuse of B has been already exploited in the sequential version, so no particular memory hierarchy implication results from the data parallel solution. The most valuable property of this solution is the latency reduction, proportional to the parallelism degree.

As requested we study in detail the data parallel map solution.

Data parallel map implementation of P_2

The generic virtual processor has the following definition: $VP_{i,j} = \{C[i,j], A[i], B[j]\}$ for $i,j = 0, \dots, M-1$. We have M^2 virtual processors each one owning one element of C and reading one element of A and one of B. The computation is a map, as no communication is necessary between workers. Ideally, the completion time is reduced from $O(M^2)$ to $O(1)$.

Passing to the actual version each worker executes a partition of the virtual processors. Different mapping strategies are possible. By assuming a *row-based partitioning* each worker is in charge of computing $g = M/n_{opt}$ contiguous rows of C. Therefore, each worker needs a partition of size g of the first array A and *all* B. A symmetric solution can be designed if C is partitioned by columns to the workers.

The input distribution process IN performs a *scattering* of A and a *multicast* of B. Its service time is:

$$T_{S-id} = T_{scatter}\ (g) + T_{multicast}\ (M)$$

Let us suppose that the home node for each channel between the IN process and a worker is the PE of the worker. The same is supposed for each channel between a generic worker and the gather process.

We can avoid providing a precise analysis of IN service time. In fact, for both the scatter and the multicast the communication latency is $O(M)$, i.e. one order of magnitude lower than the ideal service time of a generic worker. Therefore, *we can conclude that the IN process is not a bottleneck.*

For the worker we can use the same considerations described for the sequential implementation. The worker reads the blocks of its partition of A and of all B from the C2 of the sender via C2C transfers. This penalty is negligible with respect to the internal calculation phase. All the considerations about the cache faults in C1/C2 of the sequential algorithm remain valid for the worker. The ideal service time is given by:

$$T_{w-id} = \frac{M^2}{n_{opt}}(T_F + 10\tau) = 2M^2\tau$$

As expected this is exactly the inter-arrival time to the IN process. We need to take into account the send of a partition of C from the worker to the gather process. The send latency is roughly the message copy latency:

$$T_{copy}\left(\frac{M^2}{n_{opt}}\right) = \frac{M^2}{n_{opt}}T_{tras\,m} \sim \frac{M^2}{n_{opt}}\left(\frac{L_{invalidation}}{\sigma} + \beta_{code}\right) \sim 1.125M^2\tau$$

Thanks to the data partitioning of C the send to the gather process can be overlapped with the internal calculation of a worker. In conclusion the data parallel implementation exploits the optimal parallelism degree and removes the bottleneck.

c) System evaluation

Since we are able to remove the bottleneck in P_2, no further degradation exists and the bottleneck is module P_1 that cannot be parallelized. The effective service times and the efficiency can be determined by the graph analysis and are summarized in the following table:

	P_0	P_1	P_2	P_3	Σ
Ideal service time	$1M^2\tau$	$6M^2\tau$	$2M^2\tau$	~ 0	$1M^2\tau$
Effective service time	$2M^2\tau$	$6M^2\tau$	$2M^2\tau$	$2M^2\tau$	$2M^2\tau$
Efficiency	0.5	1	1	~ 0	0.5