Written Exam in HPPS—Theory Part
Example

January 12, 2021

Preamble

This is the *example* exam set for the (estimated) 3 hour theory part in HPPS, B2-2020/21. This document consists of 14 pages excluding this preamble; make sure you have them all. Read the rest of this preamble carefully. Your submission will be graded as a whole, on the 7-point grading scale, with external censorship.

- You can answer in either Danish or English.
- Remember to write your exam number on all pages.
- You do not have to hand-in this preamble.

Expected usage of time and space

The set is divided into sub-parts that each are given a rough guiding estimate of the size in relation of the entire set. However, your exact usage of time can differ depending on prior knowledge and skill.

Furthermore, all questions includes formatted space (lines, figures, tables, etc.) for in-line answers. Please use these as much as possible. The available spaces are intended to be large enough to include a satisfactory answer of the question; thus, full answers of the question does not necessarily use all available space.
If you find yourself in a position where you need more space or have to redo (partly) an answer to a question, continue on the backside of a paper or write on a separate sheet of paper. Ensure that the question number is included and that you in the in-lined answer space refers to it; e.g. write "*The [rest of this] answer is written on backside of/in appended page XX.*"

For the true/false and multiple-choice questions with one right answer give only one clearly marked answer. If more answers are given, it will be interpreted as incorrectly answered. Thus, if you change your answer, make sure that this shows clearly.

Exam Policy

This is an *individual*, open-book exam. You may use the course book, notes and any documents printed or stored on your computer and other device, but you may not search the Internet or communicate with others to answer the exam.

Errors and Ambiguities

In the event of errors or ambiguities in the exam text, you are expected to state your assumptions as to the intended meaning in your answer. Some ambiguities may be intentional.

1 Data representation and programming languages (about 25 %)

1.1 Numeric representations (about 10 %)

Numeric representations, 1.1.1:

Using a base-7 number system, having numbers in the range $[0-6]$, compute the following expressions:

1. $16_{b7} + 22_{b7}$
2. $555_{b7} + 123_{b7}$
3. $6_{b7} * 3_{b7}$

Numeric representations, 1.1.2:

Answer the following questions with a 16-bit floating point number format that:

- has no sign bit (i.e. can only represent positive numbers)
- uses a 6 bit *signed* exponent
- uses a 10 bit *unsigned* significand
- Use significand as a fraction of the maximum significand

1. What is the largest possible number?
2. What is the smallest possible non-zero number?
3. What is a valid bit pattern for the number 1.0?
4. Are there more than one valid bit pattern for the number 1.0? If yes, provide an example.

Numeric representations, 1.1.3:

For this question you are working with 7-bit signed and unsigned numbers. The unsigned numbers are using two's complement.

Fill in the blanks in this table.

Binary	Hex	Signed decimal	Unsigned decimal
0b001 1000	0x18	24	24
		42	42
		120	-8
0b100 1000	0x48		

1.2 Array representation (about 10 %)

Array representation, 1.2.1: In this task you must rewrite multidimensional array indexes to flat array indexes for arrays of various shapes. For each array and index, provide the flat index for when the array is in row-major order and in column-major order, respectively.

- **Array size:** 3×9
  Index: $(2, 4)$
  Row-major flat index: **Column-major flat index:**

- **Array size:** $1 \times 2 \times 3$
  Index: $(0, 1, 2)$
  Row-major flat index: **Column-major flat index:**

Array representation, 1.2.2: Consider a sequence of 10 values stored consecutively in memory:

a	b	c	d	e	f	g	h	i	j

Write all possible two-dimensional arrays that could be represented by this sequence in memory. All dimensions must be at least one, and all elements a—j must occur in the array once:

1.3 Tombstone diagrams (about 5 %)

Suppose that you have access to the following:

- an x86 machine,
- a compiler from C to x86 written in ARM
- an ARM interpreter written in Python
- a Python interpreter written in x86

Show Tombstone diagrams for the following cases:

1. How to execute a Python program:

2. How to execute a C program:

2 Caching and virtual memory (about 25 %)

2.1 Locality (about 5 %)

Consider the following program:

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    A[i*N+j] = A[i*N+j] * B[j*N];
```

Does it exhibit temporal and/or spatial locality? Can you optimise it to exhibit better locality?

2.2 Caches (about 10 %)

Assume a byte-addressed machine with 8-bit addresses. The machine is equipped with a single L1-cache that is direct mapped and write-allocate, with a block size of 8 bytes. Total size of the data cache is 16 bytes.

Caches, 2.2.1: For each bit in the table below, indicate which bits of the address would be used for

- block offset (denote it with `O`),
- set index (denote it with `S`), and
- cache tag (denote it with `T`).

7	6	5	4	3	2	1	0

Caches, 2.2.2: Consider we are running the following matrix transpose function, transposing a 2×2 array, on the machine above:

```
void transpose(int dst[2][2], int src[2][2]) {
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
      dst[i][j] = src[j][i];
    }
  }
}
```

It is furthermore given that:

- The `src` array starts at address 0x40 and the `dst` array starts at address 0x50.
- Accesses to the `src` and `dst` arrays are the only sources of read and write misses, respectively. `i` and `j` are allocated in registers.
- The cache is initially cold and uses LRU-replacement.

For each element `row` and `col`, indicate whether each access to `src[row][col]` and `dst[row][col]` is a hit (`h`) or a miss (`m`). For example, reading `src[0][0]` is a miss as the cache is cold. Possibly explain your answer.

`dst` array		
	col 0	col 1
row 0		
row 1		

`src` array		
	col 0	col 1
row 0	`m`	
row 1		

2.3 Virtual Memory (about 10 %)

Virtual Memory, 2.3.1: Consider a system with the following properties:

- Memory is byte-addressed.
- Virtual addresses are 15 bits wide.
- Physical addresses are 14 bits wide.
- The page size is 32 bytes.
- The TLB is 3-way set associative with four sets and 12 total entries. Its initial contents are:

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	07	0	09	0D	0	11	51	0
1	07	2D	1	12	51	1	00	00	1
2	0A	00	0	09	01	1	10	34	1
3	10	A1	0	03	0D	0	10	A0	1

- The page table contains 12 PTEs:

VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid
12	12	1	01	02	1	02	01	1	31	33	1
41	01	1	09	17	1	02	33	0	0B	0D	0
00	00	1	10	21	0	13	32	0	21	43	1

Note that all addresses are given in hexadecimal. In the following questions, you are asked, for various virtual addresses, to show the translation from virtual to physical addresses in the memory system just described. *Hint: there is one TLB hit, one page table hit, and one page fault (not necessarily in that order). This should help you double-check your work.*

Virtual address: `0x0019`

1. Bits of virtual address

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. Address translation

Parameter	Value
VPN	
TLB index	
TLB tag	
TLB hit? (Y/N)	
Page fault? (Y/N)	
PPN	

3. Bits of phys. (if any)

13	12	11	10	9	8	7	6	5	4	3	2	1	0

Virtual address: `0x0853`

1. Bits of virtual address

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. Address translation

Parameter	Value
VPN	
TLB index	
TLB tag	
TLB hit? (Y/N)	
Page fault? (Y/N)	
PPN	

3. Bits of phys. (if any)

13	12	11	10	9	8	7	6	5	4	3	2	1	0

Virtual address: `0x1337`

1. Bits of virtual address

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. Address translation

Parameter	Value
VPN	
TLB index	
TLB tag	
TLB hit? (Y/N)	
Page fault? (Y/N)	
PPN	

3. Bits of phys. (if any)

13	12	11	10	9	8	7	6	5	4	3	2	1	0

3 Computer networks (about 25 %)

3.1 Network properties (about 10 %)

Network properties, 3.1.1:

With the *network* layer from the OSI stack, answer the following questions, specific to the internet:

1. What is the responsibility of the *network* layer?
2. What adresses are used on the *network* layer?
3. What is the maximum number of addresses possible for the *network* layer?
4. How does the *network* layer handle package loss?

Network properties, 3.1.2:

For the TCP protocol, answer the following questions:

1. How does the *TCP protocol* handle package loss?
2. What is the purpose of sequence numbers in the *TCP protocol*?
3. What mechanism in the *TCP protocol* guards against network overloads?

3.2 Programs on a network (about 15 %)

Programs on a network, 3.2.1: You are writing a program where you have concluded that for each 64-bit value transfered, you need to perform 600 floating point operations (FLOPs). The machine you are computing on, is listed as being able to compute with $10,000$ FLOPs pr second. You can send at most $2,000$ bytes in a package and the ISP guarantees a speed of $4,000$ bytes/s. Now compute the following:

1. How many FLOPs will be performed per package?
2. What is the maximum bandwidth the program can handle?
3. How many FLOPs would the machine need to be able to run the program at full bandwidth?

4 Concurrency and Parallelism (about 25 %)

4.1 Multiple Choice Questions (about 4 %)

In each of the following questions, you may put one or more answers. If needed use the lines to argue for your choices.

Multiple Choice Questions, 4.1.1: Which of the following operations are guaranteed to execute atomically in a multi-threaded program?

- ☐ **a)** `printf()`
- ☐ **b)** `write()`
- ☐ **c)** `pthread_mutex_lock()`
- ☐ **d)** `x = y` (when `x` and `y` are `int`)
- ☐ **e)** `exit(0)`
- ☐ **f)** `x++` (when `x` is `sig_atomic_t`)

4.2 Scaling with Amdahl's Law (about 3 %)

For this question we use Amdahl's Law to estimate speedup in latency. Suppose we have a program where 95% of the work is parallelisable. Assuming the rest can be fully parallelised without any overhead. Answer the following:

1. What is the speedup if we run it on a 4-processor machine?
2. What about with 128 processors?
3. What is the smallest number of processors that will give us a speedup of at least 8?
4. What is the highest achievable speedup, given an arbitrary number of processors?

4.3 Scaling with Gustafson's Law (about 3 %)

For this question we use Gustafson's Law to estimate speedup in latency. Suppose we have a program where 95% of the work is parallelisable. Assuming the rest can be fully parallelised without any overhead, and that the parallel workload is proportional to the amount of processors/threads we use:

1. What is the speedup if we run it on a 4-processor machine?

2. What about with 128 processors?

3. What is the smallest number of processors that will give us a speedup of at least 5?

4. What is the smallest number of processors that will give us a speedup of at least 30?

4.4 Parallelisation (about 15 %)

The following questions are about the C-like pseudocode below:

```
float A[2*M];
for (int i = 0; i < N; i++) {
    A[0] = N;
    for (int k = 1; k < 2*M; k++) {
        A[k] = sqrt(A[k-1] * i * k);
    }
    for (int j = 0; j < M; j++) {
        B[i+1, j+1] = B[i, j] * A[2*j  ];
        C[i,   j+1] = C[i, j] * A[2*j+1];
    }
}
```

4.4.1 (about 2 %)

Explain why in the code above, neither the outer loop (of index `i`) nor the inner loops (of indices `k` and `j`) are parallel.

4.4.2 (about 3 %)

Explain why is it safe to privatise array A and show the code after privatisation.

4.4.3 (about 5 %)

Once privatised, explain why is it safe to distribute the outermost loop across the `A[0] = N;` statement and across the other two inner loops. Perform the loop distribution and show the result, while remembering to perform array expansion for `A`.

4.4.4 (about 5 %)

Use direction vectors to explain which loops in the resulting three loop nests are parallel.