NTNU
Norwegian University of
Science and Technology

Department of Mathematical Sciences

Examination paper for
TMA4280 Introduction to Supercomputing

Grading aid

Academic contact during examination: Aurélien Larcher

Phone: 45674644

Examination date: 1. June 2017

Examination time (from–to): 09:00–13:00

Permitted examination support material: B: All printed and handwritten aids permitted. Specific, simple calculator permitted.

Other information:
The examination is divided into three problems and evaluated on a total of 60 points. All the answers should be motivated:

1. Calculatory results should always be supported by a proof with intermediate steps.
2. Whenever specified in the question, interpretation of the results counts as part of the answer.
3. Specify any assumption made to support the answer.

Incomplete answers will not receive full points.

Language: English

Number of pages: 9

Number of pages enclosed: 0

Checked by:

Informasjon om trykking av eksamensoppgave	
Originalen er:	
1-sidig ☐	**2-sidig** ☒
sort/hvit ☐	**farger** ☒
skal ha flervalgskjema ☐	

Date Signature

Problem 1 Conversion and parallelization of a computational code *(22 points).* The following problem focuses on questions arising when porting a serial computational code written in C which uses single-precision storage for real numbers. The development platform is a cluster of compute nodes with a topology described in Figure 1: they are based on 64-bit Intel E5-2630v4 processors with a clock frequency of 2.2GHz, for which characteristics are given in Table 1.

The considered code involves the use of an approximation of π defined as a macro:

```
#define PI 3.141592653589
```

a) *(2 points)* Explain how real numbers are stored using the IEEE 754 single-precision binary floating-point format.

Grading aid Refer to Lecture Notes page 4, 1.2 Floating point representation: sign bit, exponent, bias, and fractional digits should be mentioned. △

b) *(2 points)* Describe the steps to obtain the binary representation of the number `PI` with a single-precision floating-point format and give the representation. Can this number `PI` be represented exactly with the given precision?

Grading aid Full points were given if the correct steps were provided and the number of 7 digits of accuracy: conversion to binary base by expansion in power of two, setting the sign bit, the exponent, then how to store the fractional digits, Lecture Notes page 5. △

c) *(2 points)* The code, initially compiled and executed as a 32-bit binary on the old machine, will now be compiled as a 64-bit binary. What are the advantages and difficulties related to such change?

Grading aid Possible advantages discussed were use of larger arrays due to 64-bit points, faster execution on native 64-bit processors, generation of code using modern instruction sets, while difficulties consist of porting to 64-bit, recompiling dependencies, and possibly larger memory requirements. △

d) *(2 points)* The code should be converted to use double-precision floating-point numbers. Describe the potential difficulties to be encountered.

Grading aid Difficulties to foresee are conversion of code to use the type *double* instead of *float*, different truncation error leading to potential change in results (which may

impact regression testing and the need for new datasets), and need to review the code for hardcoded values like the provided PI macro. △

The code involves a section (S) performing:

1. an $AXPY$ operation on two real vectors, which size may be increased to improve accuracy,
2. then summing all the entries of the resulting vector.

e) *(2 points)* What is the number of floating-point operations to perform for $AXPY$ given two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and a scalar $\alpha \in \mathbb{R}$? Give the definition of Flops and calculate the expected optimal operation rate for $AXPY$ in Flops.

Grading aid Refer to Lecture Notes page 59, 6.3 Vector–vector operations, the number of operations is $2n$. The definition of Flops is the number of Floating-point operations per seconds. Each entry in the vector involves two floating point operations, two loads, and one store. Assuming that the processor is superscalar, pipelined, and that one load per clock cycle is possible, the operation rate is guided by the fact that one can perform two floating-point operations every two memory loads. Full points were given if the correct count was provided. △

f) *(2 points)* Describe the characteristics provided by the vendor in Table 1 and explain how they are important for performance. Estimate the performance penalty in cycles for loading data from the main memory.

Grading aid Main characteristics to discuss were: multicore, superscalar, pipelining, Out-of-order and Speculative execution, SIMD instructions, large L3 cache. △

g) *(2 points)* Compute the amount of double-precision numbers that can be stored in each cache level. How does it affect the performance?

Grading aid Given that a double precision number is stored on 8 bytes, for each level a rough estimate per cache level per core consists of computing the number of doubles that can fit in the cache. △

The new implementation should be parallelized to take advantage of modern architectures.

h) *(4 points)* Which memory architecture is implemented in this compute node? Justify. Cite three ways to take advantage of parallelism provided by the hardware architecture described in Figure 1.

Grading aid The compute node is a NUMA machine consisting of two NUMA nodes, one for each socket. Each processor is a multicore with a large shared L3 cache, and has an affinity with the memory banks of the NUMA node (24GB). Each core can execute two threads simultaneously since it offers two PUs (Processing Units). Other Instruction-Level Parallelism features were also accepted. △

i) *(4 points)* Describe briefly how you would implement an hybrid MPI/OpenMP version of the code section (S) (data distribution, work-sharing, reduction), explain the functions and directives you would use. How would you chose the parameters for running on the cluster?

Grading aid Full points were given to answers providing initialization of the MPI environment, defining a proper load-balanced vector distribution, loop-parallelization with OpenMP of AXPY and partial sums, and a reduction of the partial sums with MPI, in a similar fashion as Project I. △

Architecture	4-issue pipelined superscalar processor Out-of-Order execution, Speculative execution
L1 Data Cache	32 KB per core, 64 B/line, 8-WAY, Write-back policy latency 4-5 cycles
L1 Instruction Cache	32 KB per core, 64 B/line, 8-WAY
L2 Cache	256 KB per core, 64 B/line, 8-WAY, Write-back policy latency 12 cycles
L3 Cache	25 MB, 64 B/line, 16-WAY, Write-back policy latency 59 cycles
RAM	latency 59 cycles + 46 ns
SIMD Extensions	SSE SSE2 SSE3 SSSE3 SSE4 SSE4.1 SSE4.2 AVX AVX2

Table 1: Specifications for the Broadwell processor architecture.

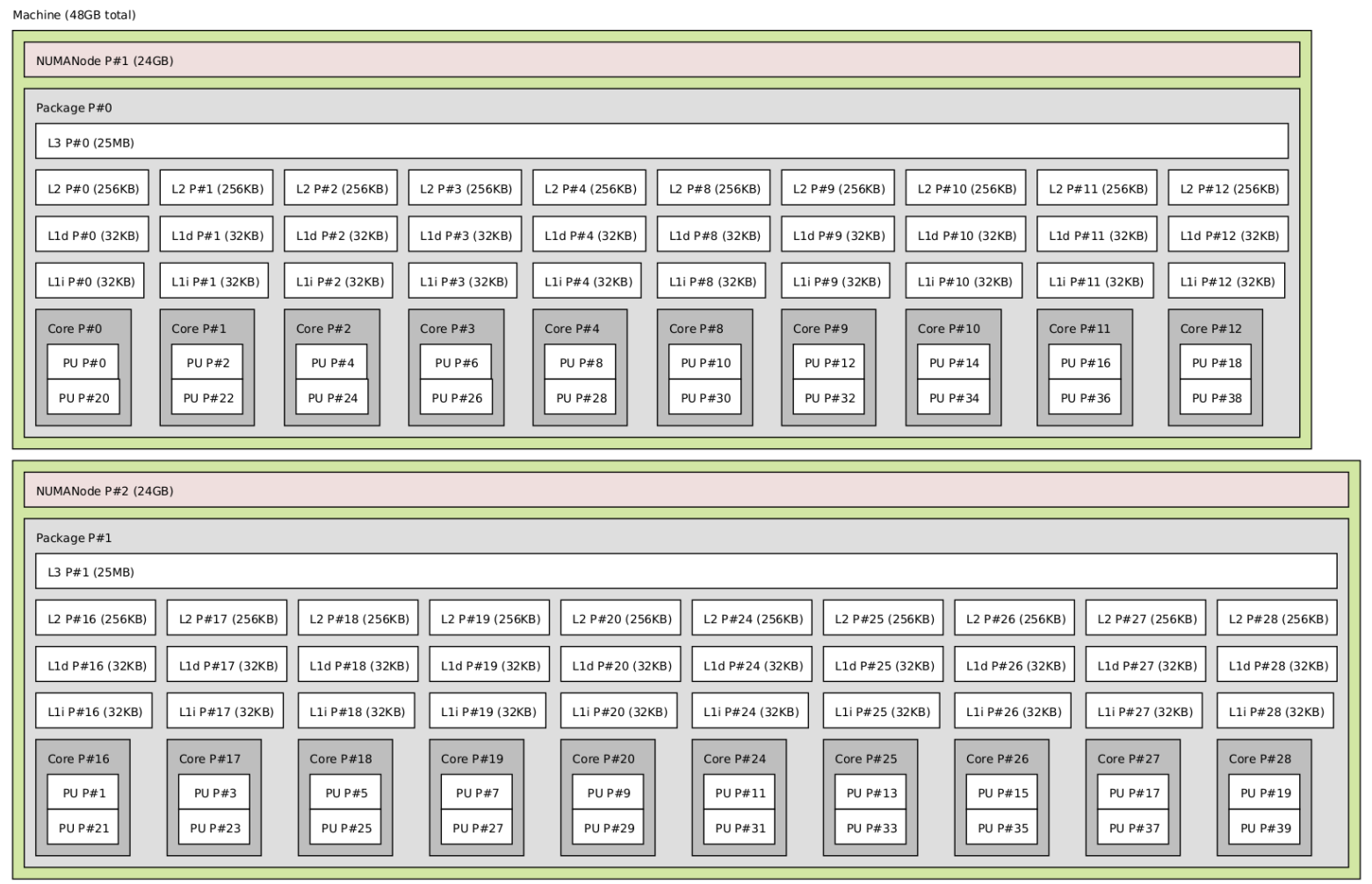


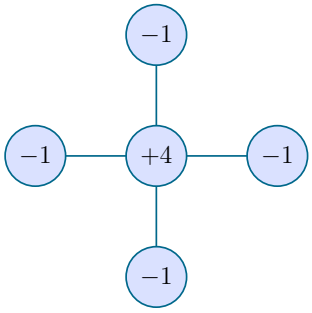
Figure 1: Topology of a dual-socket E5-2630v4 (Broadwell) compute node.

Problem 2 Solving the Poisson equation *(22 points).* Consider the solution u to the two-dimensional Poisson problem with homogeneous Dirichlet boundary conditions,

$$-\Delta u(\boldsymbol{x}) = f(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Omega = (0,1) \times (0,1), \tag{1}$$

$$u(\boldsymbol{x}) = 0 \qquad \boldsymbol{x} \in \partial\Omega. \tag{2}$$

with f a given right-hand side. Problem (1)–(2) is approximated by a finite difference method on a regular grid with $(n+1)$ points in each spatial direction, with the grid size $h = 1/n$. The Laplace operator is discretized by the standard 5-point stencil,



a) *(2 points)* Discuss the nature of the linear system. How would you implement it using PETSc with parallel performance in mind (both in terms of memory consumption and speed)?

Grading aid The matrix is sparse and should be initialized in PETSc using the function `MatMPIAIJSetPreallocation` to store only non-zero entries, and possibly setting the column indices, Lecture Slides 10. △

Instead of solving the linear system using the Fast-Diagonalization method parallelized in Project II, a Conjugate Gradient method summarized in Table 2 is used.

b) *(2 points)* Give two arguments why the Conjugate Gradient is used as an iterative method despite being intrinsically a direct method.

Grading aid The Conjugate Gradient is intrinsically a direct method since it completes after computing the basis i.e. after $N = (n-1)^2$ steps, but the convergence is slow, and the orthogonalization of residual accumulates round-off errors, Lecture Slides 9. △

c) *(4 points)* Describe the different operations involved in the algorithm, then give an estimate of the computational complexity (count of floating-point operations) and the memory requirements for one iteration.

Grading aid Operations involved are: one sparse matrix-vector product, two inner-products, and three scalar-vector multiplications + vector-vector additions. Refer to Chapter 6. Basic linear algebra performance of the Lecture Notes for the complexity and memory requirements. △

An implementation is now considered for distributed memory architectures and with a double-precision storage. Let us assume that the computational domain is partitioned into p slices of size $n \times n/p$.

The operation rate of the processor is η Flops, and the communication cost through the network interconnect is modelled with a linear relation where the time to send a message of k bytes can be expressed as

$$\tau_c(k) = \tau_s + \gamma k$$

where τ_s is the startup time (latency) and γ is the inverse bandwidth. Let us consider in particular the inner-product operation $\sigma = \boldsymbol{x}^T \boldsymbol{y}$ for two real vectors.

d) *(2 points)* Describe how the inner-product operation can be parallelized in the context of a distributed memory programming model, for a number of processors which is assumed to be a power of two. What is the computational cost for a serial run? What is the communication cost T_c of one inner-product?

Grading aid The inner-product was discussed in Lecture Slides 5: the serial run consists of $2N - 1$ floating-point operations, thus $T_1 \approx 2N\eta^{-1}$, and the communication pattern involved is a global reduction, this $T_c \approx \tau_s \log_2(p)$. △

Let us consider that the time T_p for a parallel run on $p > 1$ processes can be divided into two parts:

1. a purely parallel runtime,
2. a given communication cost.

e) *(4 points)* Express the speed-up S_p for the inner-product operation in terms of p, η, n and parameters introduced by the communication model.

Grading aid Full points were given to answers deriving the expression:

$$S_p = \frac{p}{1 + p\frac{\tau_s \log_2(p)\eta}{2N}}$$

with $N = (n-1)^2$ or equivalent, provided that justification was correct. △

f) *(4 points)* Describe the parallelization of the matrix-vector product with a one-dimensional (slice) distribution, then give an estimate of the communication cost. Use the previous results to express the speed-up S_p for one Conjugate Gradient iteration.

Grading aid Full points were given to answers introducing a partitioning of the matrix row-by-row, describing the matrix-vector multiplication by inner-products or by AXPY, then deriving a communication cost equivalent to the communication of an interprocess boundary of size $O(n)$: $T_c = 2(\tau_s + 8\gamma n)$. The serial walltime corresponding to one sparse matrix-vector product, two inner-products, and three scalar-vector multiplications + vector-vector additions, is $T_1 \approx 19n^2$ (or $T_1 \approx 15n^2$ as Examination 2004 Problem 2.a states if multiplication by one is skipped for off-diagonal terms). The total communication cost for one Conjugate Gradient iteration, corresponds to one matrix-vector product and two inner-products. The expression of S_p follows directly. △

As of November 2016 the K Computer, based on the SPARC64 architecture and located at RIKEN in Japan, was ranked number 7 for the High-Performance LINPACK dense linear algebra benchmark with 10.510 Petaflops, and was ranked number 1 with 0.6027 Petaflops for the High-Performance Conjugate Gradients benchmark, for a theoretical peak performance of 11.280 Petaflops.

g) *(2 points)* Compute the computational efficiency for both benchmarks. How do you explain the difference of efficiency between these two benchmarks?

Grading aid The computational efficiency $\epsilon = R_{max}/R_{peak}$ is 93.2% for LINPACK, and 5.3% for HPCG. The difference is explain by the fact that dense linear algebra has a higher data re-use as discussed in Lecture Notes, Chapter 6. △

h) *(2 points)* Given the rankings for both benchmarks, what can you conclude about the characteristics of the K Computer? Is this platform suitable for solving PDEs like the Poisson equation?

Grading aid The K Computer manages to top the ranking for HPCG while being only number 7 for the LINPACK benchmark, which suggests that it is better able to feed processors with data: the memory architecture, caching capabilities, and the network interconnect could be cited as possible influential factors. These results show a good balance of the supercomputer architecture. △

Given $\boldsymbol{b} \in \mathbb{R}^N$, $\mathrm{A} \in \mathbb{R}^{N\times N}$ symmetric positive definite and $\epsilon \in \mathbb{R}$ a tolerance, compute an approximate to $\boldsymbol{x} \in \mathbb{R}^N$ satistfying $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$.

Let $\boldsymbol{r}_0 = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}^0$ be the initial residual and let us assume that $\hat{\boldsymbol{x}}^0 = 0$.

While $\sqrt{\varrho_k} = \|\boldsymbol{r}_k\|_2 > \epsilon\|\boldsymbol{b}\|_2$:

1. Compute the projection factor:
$$\beta_0 = 0; \quad \beta_k = \frac{\varrho_k}{\varrho_{k-1}}, \quad \forall k > 0$$
2. Compute the direction:
$$\boldsymbol{e}_1 = \boldsymbol{r}_0; \quad \boldsymbol{e}_{k+1} = \boldsymbol{r}_k + \beta_k \boldsymbol{e}_k, \quad \forall k > 0$$
3. Compute the direction factor:
$$\boldsymbol{w} = \mathrm{A}\boldsymbol{e}_{k+1}; \quad \alpha_{k+1} = \frac{\varrho_k}{\boldsymbol{e}_{k+1}^T \boldsymbol{w}}$$
4. Update the solution:
$$\hat{\boldsymbol{x}}^{k+1} = \hat{\boldsymbol{x}}^k + \alpha_{k+1}\boldsymbol{e}_{k+1}$$
5. Update the residual:
$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \alpha_{k+1}\boldsymbol{w}$$

Table 2: Conjugate Gradient algorithm

Problem 3 General knowledge *(16 points)*

Answer by true or false, and provide a justification:

a) *(2 points)* A program with a serial fraction of $1/k$ (in time) can achieve at most a speed-up of k.

Grading aid True. Justification using Amdahl's Law was expected. △

b) *(2 points)* The computational efficiency is higher for BLAS Level 3 operations than for Level 1 operations.

Grading aid True. Refer to Chapter 6 of the Lecture Notes, specifically the discussion about data re-use. △

c) *(2 points)* Usage of `MPI_Send` and `MPI_Recv` should always be enclosed between `MPI_Barrier` calls to avoid deadlock.

Grading aid False. Firstly, these are already blocking functions, and secondly deadlock occurs because of incorrectly logic in the communication pattern. △

d) *(2 points)* In MPI only All-to-All operations are collective operations.

Grading aid False. One-to-All (broadcast, scatter) and All-to-One (reduce, gather) are also collective. △

e) *(2 points)* Maximizing the number of OpenMP threads per compute node is always optimal for performance.

Grading aid False. As seen in the Projects there is an optimal number of threads corresponding to the nature of the work involved (creating threads bears a cost), and affected by the hardware on the compute node. Answers discussing the nature of fine-grained versus coarse-grained parallelism received full points. △

Open question:

f) *(6 points)* Describe briefly the two programming models introduced during the course, and their relation to the notion of fine- and coarse-grained parallelism.

Grading aid For each programming model, one point was given to a correct definition and corresponding memory architecture, one point to the link to fine- and coarse-grained parallelism, and finally half a point to any of the following notion: message-passing, communication patterns, derived datatypes, deadlock for MPI, and loop-parallelization, scheduling, critical sections, mutexes for OpenMP. △