Fachgebiet Scientific Computing ***www.sc.tu-darmstadt.de***
Bischof, Mariano, Burger, Iwainsky, Berhe

Programmierung paralleler Rechnerarchitekturen, Winter 2014/15

Exercise 1

Solution 1.1 (Memory centric runtime analysis)**.** In this exercise we shall investigate the runtime behaviour of the computation $\mathrm{B} = \mathrm{B} + \mathrm{A}^T$ for $\mathrm{A}, \mathrm{B} \in \mathbb{R}^{n\times n}$. Assume that these matrices are so large that they do not fit in the cache. Start by considering the following algorithm for computing the sum $\mathrm{B} = \mathrm{B} + \mathrm{A}^T$:

```
1   for (i=0; i<n; i++)
2       for(j=0; j<n; j++)
3           B(i,j) = B(i,j) + A(j,i);
```

a) Name at least two main differences between CPU Cache and main memory. Explain the term cache line.

Solution: Faster access to the cache than to the main memory. The cache is smaller than the main memory. When a data item is read from main memory, l consecutive items are loaded into the cache.

b) State where in the above algorithm the memory is accessed and whether accesses are consecutive, given that the matrices A and B are stored row-wise.

Solution: In line 3 A(j,i) is read, non consecutively. In line 3 B(i,j) is read, consecutively. In line 3 B(i,j) is written, consecutively.

c) Conduct a runtime analysis of the above algorithm, expressing the runtime in terms of the following quantities:

n**,** dimension of the matrix,

T_A**,** time for the executing an arithmetic operation,

T_M**,** time for an access to the main memory,

T_C**,** time for an access to cache, and

l**,** length of a cache line.

Hint: Assume that variable values are written back to the cache.

Solution: Line is run n^2 times, this results in the following times for the above memory accesses:

Read access to A(j,i), non consecutively (from main memory):

$$T_1(n) = n^2 T_M.$$

Read access to B(i,j), consecutively:

$$T_2(n) = n^2 \left(\frac{1}{l}T_M + \left(1 - \frac{1}{l}\right) T_C\right).$$

Write access to B(i,j):

$$T_3(n) = n^2 T_C$$

Arithmetic operations:

$$T_4(n) = n^2 T_A$$

In total the runtime is

$$T^{\text{row,row}}(n) = T_1(n)+T_2(n)+T_3(n)+T_4(n) = n^2\left[T_M\left(1+\frac{1}{l}\right)+T_C\left(\left(1-\frac{1}{l}\right)+1\right)+T_A\right].$$

d) What is the runtime when the matrix A is stored column wise instead of row wise?

Solution: In this case the access to A(j,i) is consecutive, too.

In the runtime analysis in c) only the time $T_1(n)$ changes to

$$\tilde{T}_1(n) = n^2\left(\frac{1}{l}T_M + \left(1-\frac{1}{l}\right)T_C\right)$$

resulting in a total runtime of

$$T^{\text{col,row}}(n) = \tilde{T}_1(n) + T_2(n) + T_3(n) + T_4 = n^2\left[2\frac{1}{l}T_M + T_C\left(2\left(1-\frac{1}{l}\right)+1\right)+T_A\right]$$

e) Assume the following relations hold:

$$\begin{aligned} T_M &= 80\,T_A \\ T_C &= 8\,T_A \\ l &= 8. \end{aligned}$$

How many times faster is the above algorithm when you store the matrix A column wise instead of row wise?

Solution: In terms of T_A and n the algorithm requires (row wise storage of A):

$$\begin{aligned} T^{\text{row,row}}(n) &= n^2\left[80\,T_A\left(1+\frac{1}{8}\right)+8\,T_A\left(\left(1-\frac{1}{8}\right)+1\right)+T_A\right] \\ &= n^2(90\,T_A + 15\,T_A + T_A) = 106\,n^2 T_A. \end{aligned}$$

When the matrix A is stored column wise, the runtime is:

$$T^{\text{col,row}}(n) = n^2\left[2\cdot\frac{1}{8}\cdot 80\,T_A + 8\,T_A\left(2\left(1-\frac{1}{8}\right)+1\right)+T_A\right] = n^2(20\,T_A+22\,T_A+T_A) = 43\,n^2 T_A.$$

Hence the speedup is

$$S = \frac{T^{\text{row,row}}(n)}{T^{\text{row,col}}(n)} = \frac{106\,n^2 T_A}{43\,n^2 T_A} = \frac{106}{43}.$$

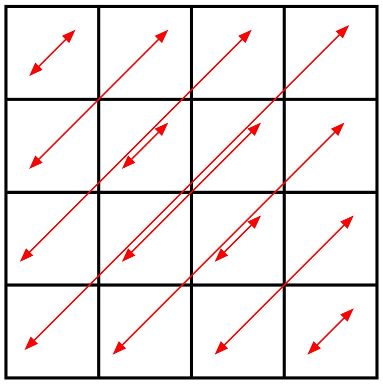

Figure 1.1: Sketch of block transposed addition algorithm.

f) Develop in broad terms a cache efficient algorithm for the computation of $B = B + A^T$, when the matrices A and B are both stored row-wise.

   Solution: For this end a block algorithm can be employed.

   This can be structured as shown in Figure 1.1:

   At any one time only one block each of A and B is accessed, so that both fit in the cache. The arrows indicate which blocks have to be present in the cache at the same time.

Solution 1.2 (C Programming)**.** In this exercise we will consider a numerical program developed by Anton Petrov and learn how to compile, link, and debug it.
Consider the c program jacobi. It computes the singular value decomposition (SVD) of a matrix $A \in \mathbb{R}^{m \times n}$. The SVD determines three matrices $U \in \mathbb{R}^{m \times m}$, $S \in \mathbb{R}^{m \times n}$, and $V \in \mathbb{R}^{n \times n}$ such that $A = USV^T$. Since U and V are unitary, it also holds that $S = U^T AV$.
Structure of the code

1) *main.c* contains the main program and has two tasks
   - create the matrices, it uses getLine
   - call the jacobi function
2) *getLine.c* is used to read lines from the input-matrix.txt
3) *jacobi.c* contains the function which implements the SVD
4) .h files contain the function declarations and the .c files contain the implementation of these functions

Perform the following tasks:

a) Compile the file *jacobi.c*. What is a suitable correct command? Which files are produced in this step?

   Solution: `gcc -Wall -c jacobi.c` produces the files jacobi.o

b) Give a Linux command which shows the recently created files in the current directory.

   Solution: `ls -lrt` shows the files in the current directory in reverse order of file file, i.e. the most recently created ones at the bottom.

c) Compile the file *getLine.c*. What is a suitable correct command? Which files are produced in this step?

   Solution: `gcc -Wall -c getLine.c` produces the files getLine.o

d) Compile the main program *main.c* and link it with the created object files. What is a suitable command? Which files are produced in this step?

   Solution: `gcc -Wall main.c jacobi.o getLine.o -lm` produces the (executable) file *a.out*
   `gcc -Wall -o program main.c jacobi.o getLine.o -lm` produces the (executable) file *program*

e) Write a *Makefile*, which helps to automate the above steps. The Makefile should have a default target that builds the program (by running the command `make`) and a target `clean` which removes the created files (by running the command `make clean`). Also, generated files should be recreated when the source file is changed. Alternatively, use an *integrated development environment* (IDE) of your choice.

   Solution:

```
# CC is the variable for c compiler
CC = gcc

# CFLAGS is the variable for the options to c compiler
CFLAGS = -g -O0 -Wall

# LIBS is the variable for libraries to be linked
# the SVD program uses c math library -lm
LIBS= -lm

# define the executable file
# the first target is the default it is executed if no target is specified
all: clean program

# executable file program depends on files main.o getLine.o jacobi.o
# automatic varibales
# $@ the name on the left side of :
#  $< the first item in the dependencies list
#  $@ and $^ are the left and right sides of the :
program: main.o getLine.o jacobi.o
	${CC} ${CFLAGS} -o $@ $^ ${LIBS}

# Generating object files
# object file main.o depends on file main.c
main.o: main.c
	${CC} ${CFLAGS} -c $< -o $@
# object file getLine.o depends on file getLine.c
getLine.o: getLine.c
	${CC} ${CFLAGS} -c $< -o $@
```

```
#object file jacobi.o depends on file jacobi.c
jacobi.o: jacobi.c
        ${CC} ${CFLAGS} -c $< -o $@
# End generating object files

# the shorthand to generate all object files from the source files
# try by uncommenting the follwing target and
# commenting the three targets specifed above
#%.o: %.c
#       ${CC} ${CFLAGS} -c $< -o $@

# pseudo (phony) target "clean" is used to delete generated files
clean:
        /bin/rm -f *.o program *-sqrt.txt *-square_and_add.txt
```

f) Run the program. What is the correct command? Why? What do you observe?

Solution: `./a.out` executes the program.

An full path has to be specified since names of executables are only searched in the directories listed in the environment variable $PATH, and the current directory is usually not included in this list.

The program crashes.

g) Run the program in a debugger. What is sensible as a preparation? In which line does the program crash?

Solution: `gdb ./program` executes the program in the GNU debugger (GDB).

The program crashes in line jacobi.c:36.

h) Run the program in a memory checker. In which line does it report the cause of the crash?

Solution: `valgrind ./program` executes the program in the valgrind memory checker (memcheck). Memcheck is actually just one of several tools that come with valgrind.

Memcheck reports uninitialized value and invalid write at line 36, relating to memory allocated(stack allocation) in line 29.

i) There is one error in the program. Can you find and fix it?

Solution: In line 36, character pointer p is assigned a value. The pointer should have been first allocated with malloc.
$p = (char*)malloc(sizeof(char));$

Task 1.1 (Speedup and Efficiency)**.** The maximum norm of a vector $\mathbf{x} \in \mathbb{R}^n$ is defined by:

$$\|\mathbf{x}\|_\infty := \max_{i=1,\dots,n} |x_i|.$$

1. Give the definitions of speedup and efficiency.

Solution: When T^{seq} is the serial runtime and $T^{\text{par}}(p)$ the runtime on p processors, then the speedup is defined by

$$S(p) \ := \ \frac{T^{\text{seq}}}{T^{\text{par}}(p)}$$

and the efficiency by

$$E(p) \ := \ \frac{S(p)}{p} \ = \ \frac{T^{\text{seq}}}{pT^{\text{par}}(p)}.$$

2. How many steps in terms of n are required determine the maximum norm of a vector of length n? *Hint*: Count only the arithmetic operations and count the operation for the maximum of two items the same as that for an absolute value.

   Solution: compute n absolute values: $T^{\text{abs}}(n) = n$.
   comparison of the n absolute values: $T^{\text{comp}}(n) = n - 1$.
   Total steps:
   $$T(1, n) \ = \ T^{\text{abs}}(n) + T^{\text{comp}}(n) \ = \ 2n - 1.$$

3. How many steps in terms of n are required if an efficient, parallel algorithm is used with $p = n/2$ processors? *Hint*: Assume that p is a power of 2.

   Solution: Every processor has 2 numbers and computes the absolute values: $T_1(n) = 2$. Cyclic reduction with respect to the maximum: $T_2(n) = \log_2(n)$. In total
   $$T(p, n) \ = \ T_1(n) + T_2(n) = 2 + \log_2(n)$$
   steps are required.

4. Give the speedup and efficiency in terms of p.

   Solution: Speedup:
   $$S(p) = \frac{T(1, n)}{T(p, n)} = \frac{2n - 1}{2 + \log_2(n)} = \frac{4p - 1}{2 + \log_2(2p)} = \frac{4p - 1}{3 + \log_2(p)} = O\left(\frac{p}{\log_2(p)}\right)$$
   Efficiency:
   $$E(p) = \frac{S(p)}{p} = \frac{\frac{4p-1}{3+\log_2(p)}}{p} = \frac{4p - 1}{3p + p\log_2(p)} = \frac{4 - \frac{1}{p}}{3 + \log_2(p)} = O\left(\frac{1}{\log_2(p)}\right)$$

5. What are the speedup and efficiency when $p = n/\sqrt{n} = \sqrt{n}$ processors are used?

   Solution: Derivation:

   Distributing the n numbers evenly among the processors, then every processor receives $\sqrt{n}$ numbers.
   Computing $\sqrt{n}$ absolute values per processor:
   $$T_3(n) = \sqrt{n}$$

Local selection of the maximum:

$$T_4(n) = \sqrt{n} - 1$$

Cyclic reduction on $p = \sqrt{n}$ processors:

$$T_5(n) = \log_2(p) = \log_2\left(\sqrt{n}\right)$$

The total runtime is:

$$T(p,n) = T_3(n) + T_4(n) + T_5(n) = 2\sqrt{n} - 1 + \log_2\left(\sqrt{n}\right)$$

For the speedup we have

$$S(p) = \frac{T(1,n)}{T(p,n)} = \frac{2n-1}{2\sqrt{n} - 1 + \log_2\left(\sqrt{n}\right)} = \frac{2p^2-1}{2p-1+\log_2(p)} = \frac{2p - \frac{1}{p}}{2 - \frac{1}{p} + \frac{1}{p}\log_2(p)} = O\left(p\right)$$

and for the efficiency

$$E(p) = \frac{S(p)}{p} = \frac{\frac{2p^2-1}{2p-1+\log_2(p)}}{p} = \frac{2p^2-1}{2p^2 - p + p\log_2(p)} = \frac{2 - \frac{1}{p^2}}{2 - \frac{1}{p} + \frac{1}{p}\log_2(p)} = O(1).$$

Solution 1.3 (Amdahl's Law)**.** Consider an arbitrary parallel code.

a) Which speedup can be achieved with 16 processors if $\beta = 1 - \alpha = 90\%$ of the code can be perfectly parallelized?

Solution: With a perfect parallelization of $\beta = 90\%$ of the code and with $p = 16$ processors the speedup is:

$$S(p) = \frac{1}{\alpha + \frac{(1-\alpha)}{p}} \implies S(16) = \frac{1}{0.1 + \frac{0.9}{16}} = \frac{160}{25} = 6.4$$

b) How large must the parallelizable share $\beta = 1 - \alpha$ be at least in order to achieve a speedup of 10 with 16 Processors?

Solution: In order to achieve a speedup of 10, the following condition must hold:

$$\begin{aligned}
S(p) &= \frac{1}{(1-\beta) + \frac{\beta}{p}} \\
\equiv \quad \beta &= \frac{p(S(p)-1)}{(p-1)S(p)} \\
\implies \quad \beta &= \frac{16(10-1)}{(16-1)10} = \frac{16 \cdot 9}{15 \cdot 10} \\
&= \frac{8 \cdot 3}{5 \cdot 5} = \frac{24}{25} = 0.96
\end{aligned}$$

Solution 1.4 (Memory Hierarchy)**.** Consider that you have an L1 cache, L2 cache, and main memory(MM). The hit rates and hit times for each are:

- 50% hit rate, 2 cycle hit time to L1
- 70% hit rate, 15 cycle hit time to L2
- 100% hit rate, 200 cycle hit time to main memory

a) What fraction of access are serviced from L2? From main memory?

Solution:
L1 miss rate= 1-0.5=50%
Fraction serviced from L2:

$$\begin{aligned} F &= L1_{miss\_rate} * L2_{hit\_rate} \\ &= 0.5 * 0.7 = 35\% \end{aligned}$$

Fraction serviced from main memory(MM):

$$\begin{aligned} L2_{miss\_rate} &= 1 - 0.7 = 0.3 = 30\% \\ F &= L1_{miss\_rate} * L2_{miss\_rate} * MM_{hit\_rate} \\ &= 0.5 * 0.3 * 1.0 = 15\% \end{aligned}$$

b) What is the miss rate and miss time for L2 cache?

Solution:

$$\begin{aligned} L2_{miss\_rate} &= 1 - 0.7 = 0.3 = 30\% \\ L2_{miss\_time} &= MM_{hit\_rate} * MM_{hit\_time} \\ &= 1.0 * 200 = 200\ cycles \end{aligned}$$

c) What is the miss rate and miss time for L1 cache? (*Hint*: depends on previous answer).

Solution:

$$\begin{aligned} L1_{miss\_rate} &= 1 - 0.5 = 0.5 = 50\% \\ L1_{miss\_time} &= L2_{hit\_rate} * L2_{hit\_time} + L2_{miss\_rate} * L2_{miss\_time} \\ &= 0.7 * 15 + 0.3 * 200 = 70.5\ cycles \end{aligned}$$

d) What is the improvement in L1 miss time, if the main memory is improved by 10%?

Solution:
New main memory hit time:

$$MM_{hit\_time} = 200/0.9 = 180\ cycles$$

Now we re-calculate the L2 miss time

$$\begin{aligned} L2_{miss\_time} &= MM_{hit\_rate} * MM_{hit\_time} \\ &= 1.0 * 180 = 180 \ cycles \end{aligned}$$

Now with the new value of the L2 miss time, the L1 miss time is

$$\begin{aligned} L1_{miss\_time} &= L2_{hit\_rate} * L2_{hit\_time} + L2_{miss\_rate} * L2_{miss\_time} \\ &= 0.7 * 15 + 0.3 * 180 = 64.5 \ cycles \end{aligned}$$

Improvement: -6 cycles or 8.5%

e) You remove the L2 to add more L1. As a result, the new L1 hit rate is 75%. What is the improvement in L1 miss time?

Solution: New L1 miss time = main memory hit time = 200 cycles
Improvement: +129.5 cycles or approx 183.69 %

$$\begin{aligned} L1_{miss\_rate} &= 1 - 0.5 = 0.5 = 50\% \\ L1_{miss\_time} &= L2_{hit\_rate} * L2_{hit\_time} + L2_{miss\_rate} * L2_{miss\_time} \\ &= 0.7 * 15 + 0.3 * 200 = 70.5 \ cycles \end{aligned}$$

f) What is the effective access time?

Solution:

$$\begin{aligned} t_{eff} &= L1_{hit\_rate} * L1_{hit\_time} + L1_{miss\_rate} * (L2_{hit\_rate} * L2_{hit\_time} + L2_{miss\_rate} * MM_{hit\_time}) \\ &= 0.5 * 2 + 0.5 * (0.7 * 15 + 0.3 * 200) \\ &= 1 + 0.5 * (10.5 + 60) = 36.25 \ cycles \end{aligned}$$

Basically the effective access time is the cummulative average time from the highest level(fastest) to the level where the hit ratio is 100%. Sometime it is used for average memory access(assuming the hit ratio at memory is 100%). We can also say that the effective access time is about the system. We can talk of an effective access time of a given level but it would be the same as for the system. Therefore the effective access time for memory hierarch of n levels is calculated as follows:

$$t_{eff} = h_1 * t_1 + (1 - h_1)h_2t_2 + (1 - h_1)(1 - h_2)h_3t_3 + .. + (1 - h_1)(1 - h_2)...(1 - h_{n-1})t_n$$

where h_i and t_i are the hit ratio and hit time of level i.