Exam Parallel Computing, June 15, 2021

Strong and Weak Scaling

Amdahl's law can be applied in contexts other than parallel processing. Suppose that a numerical application consists of 20% floating-point and 80% integer/control operations (these are based on operation counts rather than their execution times). The execution time of a floating-point operation is three times as long as other operations. We are considering a redesign of the floating-point unit in a microprocessor to make it faster.
a. Formulate a more general version of Amdahl's law in terms of selective speed-up of a portion of a computation rather than in terms of parallel processing.

0.2FP+0.8INT (in operation counts)
T_FP = 3*T_INT
, so
T_before = (0.2*3+0.8)
T_after = (0.2*x+0.8)

Speedup = (T_FP+T_INT)/(T_FP*+T_INT)

Speedup=(0.2*3 + 0.8)/(0.2*x+0.8)

b. How much faster should the new floating-point unit be for 25% overall speed improvement?
1.25=(0.2*3 + 0.8)/(0.2*x+0.8)
1.25*(0.2*x+0.8)=0.2*3 + 0.8
0.25x+1=1.4
0.25x=0.4
x=1.6
So FP unit time goes from 3 down to 1.6 ➔ 1.875x

c. What would be the overall speedup if the new floating-point unit is able to perform its operations as fast as all other integer/control operations?
speedup=(0.2*3 + 0.8)/(0.2+0.8)
speedup=1.4

d. What is the maximum speed-up that we can hope to achieve by only modifying the floating-point unit?

Since floating point division makes up only 20% of the instructions but they take 3cycles in the original machine. We can not reduce the overall running time to less than the remaining 80% of single cycle instructions in the original machine (in fact, we cannot achieve this since we will need a floating-point division to be performed in time 0).
So, the maximum speedup would be bounded by (0.20*3+.80) / .80 = 1.75.

e) Now let us switch *back to parallel computers*. A programmer has parallelized a program such that s, the amount of time the program spends in serial code, is 1% of the overall execution time. Moreover, the problem size can expand as the number of processors increases. What is the expected speedup on 20 processors?

(*using Gustafson*):
Speedup = P + (1-P) s
= 20 – 19 * 0.01 = **19.81**

Architecture

Your friend implements the following parallel code for generating a histogram from the values in a large input array. For each element of the input array, the code uses the function bin_func to compute a “bin” the element belongs to (bin_func always returns an integer between 0 and NUM_BINS-1), and increments a count of elements in that bin. His port targets a small parallel machine with only two processors. This machine features 64-byte cache lines. Your friend’s implementation is given below.

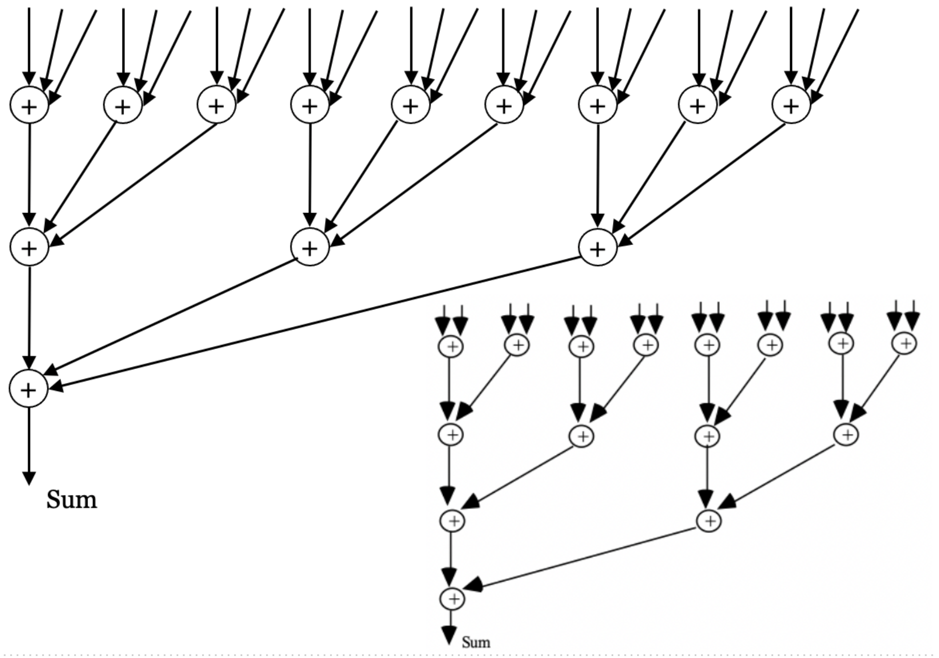
```
float input[N]; // assume input is initialized and N is very large

int histogram_bins[NUM_BINS]; // output bins
int partial_bins[2][NUM_BINS];// assume bins are initialized to 0
                              // assume partial_bins is 64-byte aligned
///////////////////////// Code executed by thread 0 /////////////////

for (int i=0; i<N/2; i++)
   partial_bins[0][bin_func(input[i])]++;
barrier(); // wait for both threads to reach this point
for (int i=0; i<NUM_BINS; i++)
   histogram_bins[i] = partial_bins[0][i] + partial_bins[1][i];
///////////////////////// Code executed by thread 1 /////////////////

for (int i=N/2; i<N; i++)
   partial_bins[1][bin_func(input[i])]++;
barrier(); // wait for both threads to reach this point
```

Your friend runs this code on an input of 1 million elements (N=1,000,000) to create a histogram with eight bins (NUM_BINS=8). He is shocked when his program obtains far less than a linear speedup, and he plans to completely restructure the code to eliminate load imbalance. You take a look and recommend that he not do any coding at all, and just create a histogram with 16 bins instead. Explain why.

Answer
With a 64-bit-wide cache line and 8 bins, the partial_bins arrays for each thread lie on the same cache line (8 integers = 32 bytes). As a result, although there is no data sharing between the two threads when computing the partial results, significant false sharing will occur. Increasing the number of bins to 16 causes, partial_bins[0] and partial_bins[1] to reside on a separate cache lines, and false sharing is eliminated. It could also be noted that there is very little load imbalance in the current solution. The threads each process 500,000 elements in parallel. Then the serial part of the code is a simple summation of eight numbers.

Q2

SciPA Technologies has decided to become the leader in the large scientific computing server market, and has developed a cutting-edge computer system named Trifecta. The Trifecta computer is unique in that it supports reduce and broadcast trees of the third degree instead of the traditional degree-two trees supported by its competitors. In addition, Trifecta also supports a fused floating-point multiply-add operation (the operation c + a ∗ b is performed within one unit of time). Naturally, the floating-point 3-inputs wide add operation (c + a + b) is also completed in one unit of time. A reduction on this network is shown in the figure below along with the traditional approach used by all competitors.



Compare the time it takes to perform a sum reduction operation over 729 processors on the Trifecta computer to the time it takes to perform the same operation on a regular computer with the same number of processors that performs a conventional 2-inputs reduce. Assume the communication between processors also takes one unit of time on both machines (as well as the 2-input additions of the conventional system). Fill the times below but also resent your work.

Answer

$\log_3 P + \log_3 P = 2 \cdot \log_3 P = 12$

$\log_2 P + \log_2 P = 2 \cdot \log_2 P = 20$

Accelerators

You are asked to program a system with a GPU accelerator that has four streaming multiprocessors each of which entails 256 ALUs (processing cores). Each ALU has a multiplier **and** an adder and can execute one floating point multiplication and one floating point addition per clock cycle. The GPU chip on the accelerator is operated at a frequency of $1.25*10^9$ Hz. What is the theoretical peak performance of this GPU accelerator? Specify your answer in bilions (10^9) of flop/s.

Answer (2,560)

SMs=4 (streaming multiprocessors)
ALUs=256
ops=2 (MUL and ADD)
f=1.25*10^9
flops=cores*alus*ops*f/10^9 = 2,560

The GPU accelerator from the previous question has a device memory. The GPU chip can communicate with the device memory at a bandwidth of 224 GB/s ($224*10^9$ Byte/s) and latency of 1 microseconds ($1*10^{-6}$ s). Data is transferred from the CPU memory to the GPU memory via the PCIe interface that has a bandwidth of 4 GB/s ($4*10^9$ Byte/s) and latency of 10 microseconds ($10*10^{-6}$ s). You are asked to accelerate matrix-vector multiplication Y = A.X on the GPU where A is a 6000x6000 matrix and X an Y are vectors of 6000 elements each. For efficiency, the elements of these arrays are half-precision floating point numbers (16-bits). For the execution of Y = A.X on the Accelerator the matrix A is first copied from the CPU memory to the device memory. How much time will this copy operation take? Specify your answer in microseconds (10^{-6} s).

Answer (18,010)

gpu_chip_memory_bw=224*10^9
gpu_chip_memory_lt=1*10^-6
cpu_gpu_bw=4*10^9
cpu_gpu_lt=10*10^-6
n=6000
(n*n*2/cpu_gpu_bw+cpu_gpu_lt)/10^-6

After copying the matrix A is completed, you should also copy the 6000 elements of vector X. Only after this is completed your intended GPU program can start the computation on the GPU accelerator. This action will obvously take some time as well. Once the computation on the accelerator is finished (we assume everything fits in the memories and all data transfers use the bandwidths at 100% efficiency) we have to copy back the result vector Y (reminder: Y is 6000, 16-bits wide elements sent using the PCIe interface at 4GB/s). How much time will this action (copying vector Y from the device to the CPU memory) take? Specify your answer in microseconds (10^{-6} s).

Answer (13)

gpu_chip_memory_bw=224*10^9
gpu_chip_memory_lt=1*10^-6
cpu_gpu_bw=4*10^9
cpu_gpu_lt=10*10^-6
n=6000
#Y from CPU to GPU
(n*2/cpu_gpu_bw+cpu_gpu_lt)/10^-6 = 13

Now by ignoring all the other details, e.g., the fact that half-precision will simply double the theoretical performance from the first question. What performance do you expect from the system just based on the times you will need to send the matrix A and vector X to the accelerator and receive back the output vector Y? (Note: the time needed to send vector X is the same as the time for vector Y in the last question. We also assume there is no overlapping of the three actions)

Provide your estimation (in bilions (10^9) of flop/s) along with your reasoning about the process and the expected results.

Answer

"total_time"18,010+13+13= 18,036 us (A and X --> and Y <---)
total_ops=n*n*2 #72 millions
total_ops/total_time*10^-9 gives **3,992 flop/s**

So if you now see that 2,560 (actually 5,120) vs 3,992 gives 33% overhead.

MPI

Q1: One or more MPI communication operations take(s) values of data owned by each process from:

```
P0: b1 [1 2 4 9 3 5 6 7 8]

P1: b1 [? ? ? ? ? ? ? ? ?]

P2: b1 [? ? ? ? ? ? ? ? ?]

P3: b1 [? ? ? ? ? ? ? ? ?]
```

To

```
P0: b1 [1 2 4 9 3 5 6 7 8]

P1: b1 [1 2 4 9 3 5 6 7 8]

P2: b1 [1 2 4 9 3 5 6 7 8]

P3: b1 [1 2 4 9 3 5 6 7 8]
```

Show the MPI communication operation(s) and any conditional (if) statements needed to cause the reordering of data gives the resulting data.

Answer

MPI_Bcast(b1, 9, MPI_INT, 0, MPI_COMM_WORLD);

Q2: One or more MPI communication operations takes values of data owned by each process from:

```
P0: b1 [0 1 2 4 9 3 5 7 6]

P1: b1 [7 6 0 1 2 4 9 3 5]

P2: b1 [3 5 7 6 0 1 2 4 9]

P3: b1 [4 9 3 5 7 6 0 1 2]
```

To:

```
P0: b1 [14 21 12 16 18 14 16 15 22]

P1: b1 [14 21 12 16 18 14 16 15 22]

P2: b1 [14 21 12 16 18 14 16 15 22]

P3: b1 [14 21 12 16 18 14 16 15 22]
```

Show the MPI communication operation(s) and any conditional (if) statements needed to cause the reordering of data gives the resulting data.

Answer

```
MPI_Allreduce(b1, b1, 9, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
```

Q3: One or more MPI communication operations takes values of data owned by each process from:

```
P0: b1 [0 6]

    b2 [? ?]

P1: b1 [2 4]

    b2 [? ?]

P2: b1 [9 3]

    b2 [? ?]

P3: b1 [5 7]

    b2 [? ?]
```

To:

```
P0: b1 [0 6]

    b2 [5 7]
```

```
P1: b1 [2 4]

    b2 [0 6]

P2: b1 [9 3]

    b2 [2 4]

P3: b1 [5 7]

    b2 [9 3]
```

Show the MPI communication operation(s) and any conditional (if) statements needed to cause the reordering of data gives the resulting data.

Answer

```
MPI\_status *s;
if (pid $ < 3 $) {
  MPI_Send(b1, 2, MPI_INT, pid+1, 0, MPI_COMM_WORLD);
} else {
  MPI_Send(b1, 2, MPI_INT, 0, 0, MPI_COMM_WORLD);
}
if (pid $ > 0 $) {
  MPI_Recv(b2, 2, MPI_INT, pid-1, 0, MPI_COMM_WORLD, s);
} else {
  MPI_Recv(b2, 2, MPI_INT, 3, 0, MPI_COMM_WORLD, s);
}
```

Q4: One or more MPI communication operations takes values of data owned by each process from:

```
P0: a [1 2 3 4]

    b [?]

P1: a [1 2 3 4]

    b [?]

P2: a [1 2 3 4]

    b [?]

P3: a [1 2 3 4]

    b [?]
```

To:

```
P0: a [1 2 3 4]
```

```
    b [1]

P1: a [1 2 3 4]

    b [2]

P2: a [1 2 3 4]

    b [3]

P3: a [1 2 3 4]

    b [4]
```

Show the MPI communication operation(s) and any conditional (if) statements needed to cause the reordering of data gives the resulting data.

Answer

MPI_Scatter(&a,4,MPI_INT,&b,1,MPI_INT,0,MPI_COMM_WORLD);

Q5: Name and describe the three main MPI programming models (paradigms).

Given a scenario where multiple tasks exist, however each task requires a (significantly) different amount of time to execute. What is the most suitable model to use? Justify your answers.

Answer

Single Program Multiple Data (SPMD): all processing nodes execute the same program.
Master-worker / server-client: one of the processing nodes executes a program which is different from the program executed by all other nodes. The master process can distribute tasks to the rest.
Each processor executes its own program.
For the given scenario, the best model would be Master-worker, since it will ensure that the workload is balanced across threads despite the tasks taking different amount of time to complete.

Q6: The following code appears to be used to synchronize processes 0 and 1 (i.e. guarantee they both executed line number 102). Will it work as expected? How can you fix it using blocking calls and without introducing a barrier?

```
100: char flag;
101: int checkpoint_tag = 102;

102: if (rank == 0) {
103:    flag = 1;
104:    MPI_Send(&flag, 1, MPI_BYTE, 1, checkpoint_tag,
MPI_COMM_WORLD);
} else if (rank == 1) {
   MPI_Recv(&flag, 1, MPI_BYTE, 0, checkpoint_tag, MPI_COMM_WORLD
, MPI_STATUS_IGNORE);
}
if (rank == 0)
   printf("Processes %i and %i executed line %i \n", 0 , 1, 102);
```

Answer

No it won't work as expected. If the MPI runtime provides an internal buffer, receiving the message cannot be ensured upon issuing a blocking send. Therefore, rank 0 could proceed before rank 1 have issued the receive. To fix that, use the synchronous MPI_Ssend.

OpenMP

Assume the following simple OpenMP code:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main()
{
   int c1 = 0;
   int c2 = 0;
   int i;
   omp_set_num_threads(4);
   #pragma omp parallel
   {
      #pragma omp atomic
      c1++;
      printf("par: %d\n", omp_get_thread_num( ));       // Line1
      #pragma omp master
      {
         #pragma omp critical
         {
         c2++;
         printf("master: %d\n", omp_get_thread_num( )); // Line2
         }
      }
      #pragma omp single
      {
         #pragma omp critical
         {
         printf("single: %d\n", omp_get_thread_num( )); // Line3
         c2++;
         }
      }
   }
   printf("c1: %d, c2: %d\n", c1, c2); // Line4
   return(0);
}
```

a) What value(s) is/are printed for the thread number(s) in Line1?

0

0 or 1 or 2 or 3

0 and 1 and 2 and 3

cannot say anything about the values, there is a race

b) What value(s) is/are printed for the thread number(s) in Line2?

0

0 or 1 or 2 or 3

0 and 1 and 2 and 3

4

c) What value(s) is/are printed for the thread number(s) in Line3?

0

0 or 1 or 2 or 3

0 and 1 and 2 and 3

something else

d) What value is printed for the c1 in Line4?

2

3

4

1 or 2 or 3 or 4

e) What value is printed for the c2 in Line4?

2

3

4

1 or 2 or 3 or 4

The following C program initialises a shared two dimensional array A of size **1000x1000** such that A[i][j]=5*i+7*j for all valid combinations of i and j, and then counts the number of elements that are evenly **divisible by 3**. The computational load of initializing the elements of A and checking if they are divisible by 3 is shared equally by 20 threads. When a certain thread identifies an element that is divisible by 3, it increments the shared variable `cnt`. At the end, the main thread also prints the value of `cnt` that is the total number of elements of A that are divisible by 3.

You are expected to augment this program with the correct OpenMP constructs/facilities starting on line 18:. If you would need to insert more OpenMP lines indicate where they will fit in the code listing, e.g, between lines 11: and 12:. Provide your argumentation for the OpenMP constructs you are using with a brief explanation of what their role is.

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include <time.h>
 4: #define N 1000
 5: int A[N][N];
 6:
 7: int divBy3(int n) {
 8:   return (n%3 == 0 ? 1 : 0);
 9: }
10:
11: int main(int argc, char **argv) {
12:   for (int i = 0; i < N; ++i) {
13:     for (int j = 0; j < N; ++j) {
14:       A[i][j] = 5*i + 7*j;
15:     }
16:   }
17:   int cnt = 0;
18:   #pragma ...............................
19:   for (int i = 0; i < N; ++i) {
20:     for (int j = 0; j < N; ++j) {
21:       cnt += divBy3(A[i][j]);
22:     }
23:   }
24:   printf("%d\n", cnt);
25:   return 0;
26: }
```

Answer

#pragma omp parallel for collapse(2) reduction(+:cnt)