Project Euler Problem 8:
Largest Product in a Series

1 Design

1.1 The Problem

Find the m adjacent digits in the n-digit number that have the greatest product.

1.2 STAPL

1.2.1 Pseudocode

Input: n, m
Output: the greatest product of any adjacent m digits in a random sequence of n digits.

```
Fill an array with $n$ random digits.
Create overlap views for every combintion of $m$ adjacent digits.
Compute the product of the digits in each overlap view.
Find the highest value.
```

1.2.2 Components

- stapl::array<int>
  - Holds the inital n random digits
  - Stores the products of each set of m adjacent digits
- stapl::array_view<stapl::array<int>>
  - Allows access to each of the stapl::array's described above
- stapl::result_of::overlap_view<stapl::array_view<stapl::array<int>>>::type
  - Creates views containing subsets of the random digits
- stapl::map_func()
  - Fills an array with random values
  - Computes the product of digits in overlap views and stores it in an array view of results
- stapl::max_value()
  - Finds the maximum value among the products
- stapl::counter<stapl::default_timer>
  - Tracks execution time of the program

- stapl::do_once()
    - Provides print output

2 Experiments

The experiments were performed on a Cray XE6m machine with 24 nodes and a total of 576 cores. The system features 2GB of memory per core. The n value varied throughout the experiments and is indicated. The m-value used was 9.

2.1 Strong Scaling

With respect to strong scaling, scalability is defined as the ratio of execution time on one processor to execution time on p processors. It is desirable for a graph of scalabilty to be linear in nature.

Table 1 shows the average execution time (seconds) for 32 runs for each combination of data set size and processor count. Table 2 shows execution time (seconds) for a series of runs which were conducted in a way which forced execution to take place on two seperate chips. This was accomplished by limiting the number of cores per chip to one half of the total number used.

	Input	
Processors	10,000,000	100,000,000
1	2.08412813	23.191062
2	1.30868375	14.34445313
4	0.701937	7.773844688
8	0.4230231563	4.78556
16	0.46476075	4.820199063
32	0.2242470313	2.527174063
64	0.1382736875	1.542254375
128	0.0767856563	0.7645958125
256	0.0420141875	0.4058763438
512	0.0268319063	0.2023741563

Table 1: This table shows average execution time per input and processor count

	Input
Processors	1,000,000
1	30.00724063
2	14.99980625
4	8.102572188
8	4.0788325
16	2.348421875
32	1.221450938

Table 2: This table shows average execution time per input and processor count when execution is forced to take place on two chips

Figures 1 and 2 show graphs of execution time for a given n and a set number of processors.

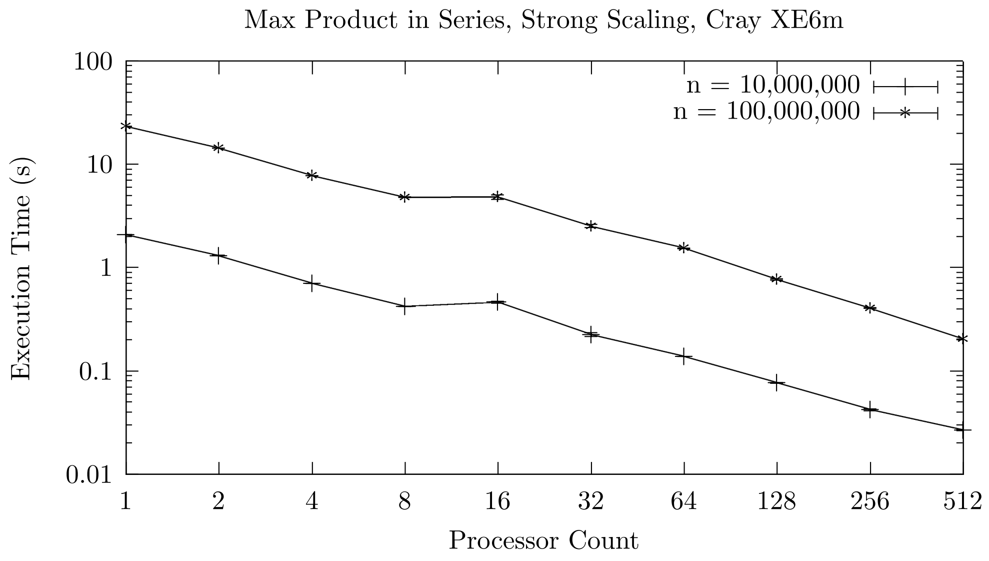

Figure 1: This graph shows average execution time per input and processor count

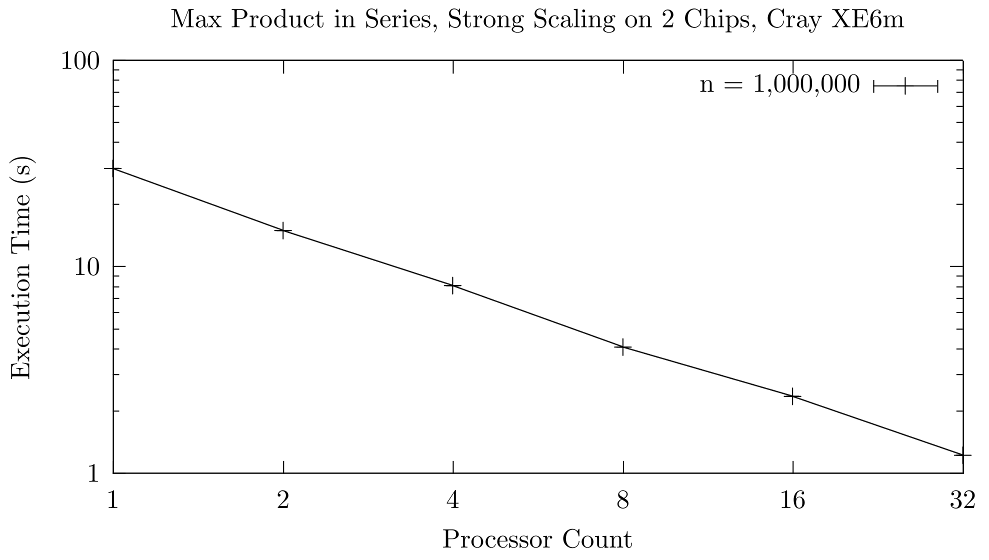

Figure 2: This graph shows average execution time per input and processor count when execution is forced to take place on two chips

2.2 Weak Scaling

Weak scaling relates to how the execution time varies with the number of processors for a fixed problem size per processor. A value close to one is indicative of a program which scales well.

	Input Size Per Processor	
Processors	1,000,000	10,000,000
1	1	1
2	1.0839433849	1.0846647847
4	1.0903083815	1.0910340174
8	1.1852665158	1.1860553493
16	1.3144923938	1.3153672315
32	1.301122399	1.3019883385
64	1.352317299	1.3532173104
128	1.3581480753	1.3590519673
256	1.3684356208	1.3696710727
512	1.3660843635	1.3669935373

Table 3: This table shows normalized execution time ($\frac{T_p}{T_1}$) for a given input size per processor and processor count

Figures 3 and 4 each show graphs of the normalized execution times given a specific input size per processor.

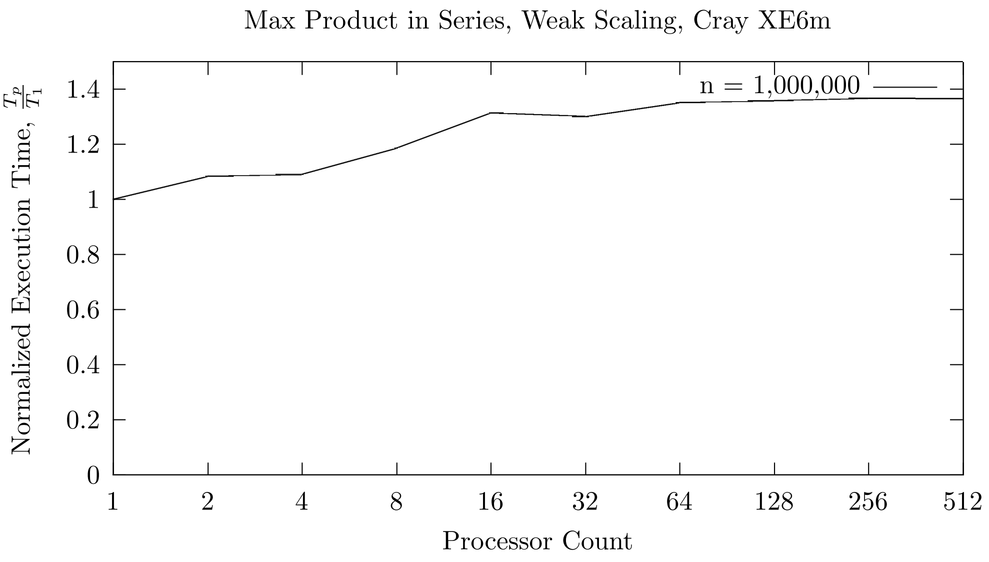

Figure 3: This graph shows normalized execution time for n=1,000,000

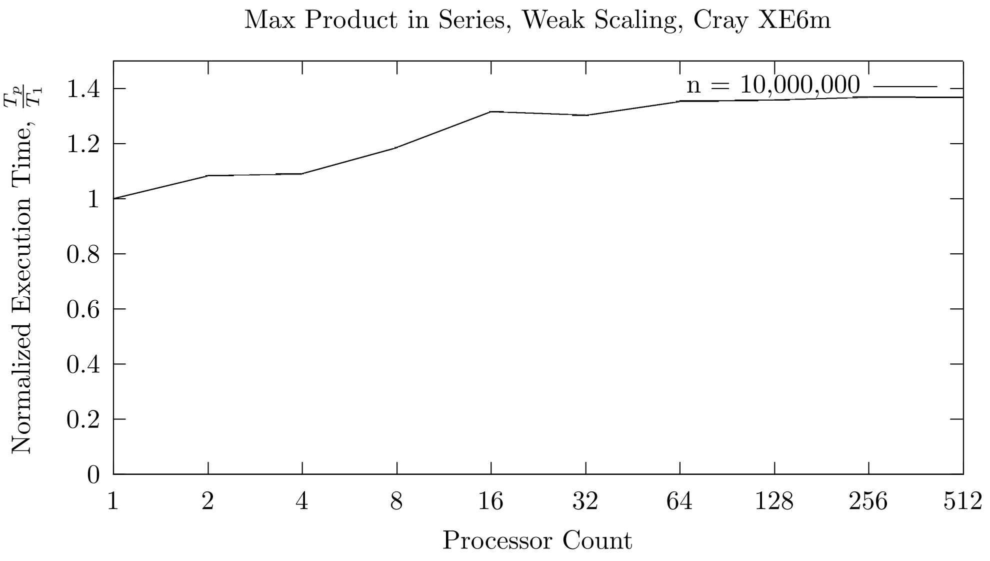

Figure 4: This graph shows normalized execution time for n=10,000,000

3 Analysis

This section discusses the results from the experiments.

3.1 Strong Scaling

Figure 1, a graph built from Table 1, shows a desireable negative linear slope for each input size. We note a slight increase in execution time at processor count = 16. Prior to the experiments we were expecting to see this increase occur when using 32 processors, as it would mark the transititon point from being able to run on a single chip with 16 cores, to requiring multiple chips, thus increasing communication. Our Cray system features 16 core AMD Opteron processors, each consisting of two banks of 8 cores with separate memory controllers. It was suspected that the increase at 16 was due to memory accesses taking place between the RAM of one bank of 8 cores and the other.

A second series of runs was conducted using the mppnppn directive to force execution to tke place on two different chips. This slowed the overall execution time, but enabled us to be sure no cross memory accesses were taking place within a chip. As illustrated in Figure 2, this approach eliminated the irregularity when the processor count was equal to 16. This supports the hypothesis that memory accesses on an individual chip, but between banks, were responsible.

3.2 Weak Scaling

In our weak scaling experiments we obtained nearly identical results for both trials. One trial involed an input size of one million per processor, and the other featured an input of ten million per processor.

In an ideal situation the program would scale nearly perfectly, meaning the execution time would be near constant as processor count and total input size increased. This program is limited by several factors, including that we need to find the highest product.

Finding the max is an operation which depends on the total input size, and is therefore limited. Other tasks, such as generating the random input, can scale much better. Our overall results were what we were expecting to see in terms of weak scaling.