Communication Improvement for the LU NAS Parallel Benchmark: A Model for Efficient Parallel Relaxation Schemes

Maurice Yarrow
Sterling Software at NASA Ames Research Center
Moffett Field, California

Rob Van der Wijngaart
MRJ Technology Solutions at NASA Ames Research Center
Moffett Field, California

Abstract

The first release of the MPI version of the LU NAS Parallel Benchmark (NPB2.0, ref.1) performed poorly compared to its companion NPB2.0 codes. The later LU release (NPB2.1 & 2.2, refs.2,3) runs up to two and a half times faster, thanks to a revised point access scheme and related communication scheme. The new scheme sends substantially fewer messages, is cache "friendly", and has a better load balance. We detail the observations and modifications that resulted in this efficiency improvement, and show that the poor behavior of the original code resulted from deriving a message passing scheme from an algorithm originally devised for a vector architecture.

Introduction

The NAS Parallel Benchmark (NPB) LU code is a simplified compressible Navier-Stokes equation solver. It employs the well-known symmetric point-Gauss-Seidel relaxation scheme for solution of the discretized, linearized equations. The natural point access method for this scheme is by row or by column. Unfortunately, such access amounts to a recurrence relation on previously calculated values, and thus constitutes an array dependency for vector architectures. The production flow solver after which LU was modeled was developed to run on vector architecture machines, and was written carefully so as not to compromise vectorization. The so-called *diagonal* access scheme, described by Venkatakrishnan et al. (see ref.1, and below), removes the recurrence, so that full advantage can be taken of the vector pipeline arithmetic units.

The processors in today's parallel machines, however, typically rely on fast intermediate caching systems to feed the cpu with a steady stream of data and instructions, since the main memories of these machines are built of inexpensive DRAM for which data access is relatively slow. Vector units are usually absent in these systems, so recurrence is not a major issue. But compromising cache behavior can greatly deteriorate performance. The continued large-stride memory access of the diagonal scheme results in frequent fetches of fresh cache lines of data from main memory, and in ejection of lines from cache before all their information has been used. The key to optimum cache behavior is the strong reuse of the data contained within cache lines (data "locality"). Column-based point relaxation schemes exhibit such locality, and are thus cache "friendly" numerical methods.

The first release of the LU MPI source code (in NPB2.0) was derived from a vector implementation, from which it inherited the diagonal relaxation access method. The three-dimensional Cartesian grid, whose physical x, y, and z coordinates are indexed with i, j, and k, respectively, is distributed using a two-dimensional domain decomposition; each processor receives a vertical partition—*pencil*—of points (k-direction is in-processor). Active points subject to relaxation are all interior grid points. Relaxation starts at an active corner point ($i = 2$, $j = 2$) of the grid on the bottommost ($k = 2$) active grid plane. The next points to be relaxed are on the diagonal consisting of only two active grid points ($i + j = 5$), also in the bottommost active plane. This is followed by the adjacent diagonal consisting of three active points ($i + j = 6$), and so on. These active diagonals sweep from one corner of the active plane towards the opposite corner. The order in which points may be relaxed is determined by the same criteria that govern a typical Gauss-Seidel relaxation process. For the LU code, the discretization stencil at a given point (i, j, k) includes, in three dimensions, the six neighbor points with indices $(i \pm 1, j, k)$, $(i, j \pm 1, k)$, and $(i, j, k \pm 1)$. In this scheme, all relaxation progress occurs on one k-plane at a time. Therefore, for this analysis we may concern ourselves with only the (i, j) indexes, as if the problem were two-dimensional. With this in mind, the Gauss-Seidel relaxation scheme with natural point ordering dictates that point (i, j) may be relaxed only after points $(i - 1, j)$ and $(i, j - 1)$ have already been updated. For the diagonal access method, we note that all points (i, j) on a given diagonal may be defined as those for which the sum $i + j$ is constant. These may be relaxed independently of each other but only after all points on the previous diagonal have been relaxed. A little thought will suffice to see that where a diagonal crosses partition boundaries, values at a neighbor point on the neighbor processor will be required. These will be the endpoints of the previously updated diagonal on the neighbor processor.

Communication in the NPB2.0 LU code occurs at each end of diagonals continued across processors, and consists of the transmission of 5 floating point values (for density, x-, y- and z-momentum, and energy) per end point. This transmission must occur before the relaxation can proceed on the next diagonal where this subsequent diagonal is continued on an adjacent processor; recall that points cannot be relaxed on any given diagonal before all points of the previous diagonal in the same plane have been relaxed, regardless of which processor owns those points. Therefore, communication must occur for each diagonal intersection with processor grid boundaries. This explains the inordinate amount of communication of the original code (up to 75% of total execution time, depending on problem size and number of processors, on an IBM SP-Wide Node). Note that when a given processor has completed all diagonals in its section of plane $k = k_0$, it proceeds to the first diagonal (which consists of a single point) in its section of the plane immediately above, i.e. $k = k_0 + 1$ (see ref.1). Once a diagonal intersects with partitions owned by processors to the "east" and/or "north" (larger i- and/or j-values, respectively), those processors can start execution concurrently. Execution of the relaxation process is functionally data parallel.

Cost Model for Diagonal Scheme

We examine performance first for a specific example of the original NPB2.0 scheme. Consider the Class A problem size (64^3 grid points), running on 64 processors. The processor grid is an 8 by 8 grid, and each processor will have an 8 point (x direction) by 8 point (y

direction) by 64 point (z direction) grid portion. This vertical pencil consists of 64 "tiles" of constant k, each measuring 8 by 8 grid points. The 15 diagonal sections within each tile vary in size from 1 to 8 points, and back to 1. Exactly 279 floating point operations (FLOPs) are required to relax each point. Assume an average cost of f seconds per FLOP, a communication latency of l seconds, and a transmission time of b seconds per byte. Note that processors with pencils in the interior of the grid must send 5 double precision words at each of the two ends of the diagonals in the "north-eastern" half of the tile. Note also that progress along diagonals in this method occurs—functionally—in lockstep on participating processors, but that the lengths of the sections of a diagonal owned by adjacent processors alternate. Clearly, adjacent sections are of the exact same length only at the $1/4$ and $3/4$ "stages" within the tile, when actual section length equals average length. Therefore, the largest diagonal section length must be the one used to determine the *effective* computation cost for relaxing points, and a significant load imbalance is necessarily incurred. Then each tile takes $88(279f) + 16(5 \cdot 8b + l)$ seconds to process. Of this time, $16(40b + l)$ seconds is the time spent on communication, and the *effective* number of points to be solved on each tile (i.e. the number compensating for the load imbalance) is 88, not 64. This is true because processors operate on a diagonal in lock-step fashion. For tiles of size 8 by 8, at least one of the involved processors will always have at least 4 points on its section of the diagonal, and up to 8 points for the largest section of diagonal. For example, a processor relaxing the first singleton point on a tile has diagonal neighbors whose sections of the same diagonal line each have 7 points to relax, so the *effective* number of points here is 7. So instead of determining computation cost from the actual number of points on the tile, which is $64 = 1 + 2 + \cdots + 7 + 8 + 7 + \cdots + 2 + 1$, we use the *effective* number of points, $88 = 7 + 6 + 5 + 4 + 5 + 6 + 7 + 8 + 7 + 6 + 5 + 4 + 5 + 6 + 7$.

More generally, consider a grid size of N^3 points and number of processors $P = 2^p$ where either $p = 2k$ or $p = 2k + 1$, $k = 1, 2, 3, \ldots$. Then each tile measures $(N/2^k \times N/2^k)$ points when $p = 2k$, or $(N/2^k \times N/2^{k+1})$ points when $p = 2k + 1$. Therefore, each tile takes

$$\left\{2\frac{N}{2^k} + 4\sum_{m=N/2^{k+1}+1}^{N/2^k-1} m\right\} \cdot 279f + 2\frac{N}{2^k} \cdot (40b + l) = \frac{N}{2^k}\left(\frac{3N}{2^{k+1}} - 1\right) \cdot 279f + \frac{N}{2^{k-1}} \cdot (40b + l)$$

seconds to process if $p = 2k$, and

$$\left(\frac{3N^2}{2^{2k+2}} - \frac{N}{2^{2k+1}}\right) \cdot 279f + \frac{3N}{2^{k+1}} \cdot (40b + l)$$

seconds to process if $p = 2k + 1$. In both cases, the expression multiplying $279f$ represents the the effective number of points on each tile to relax, and the second term is the communication cost.

Using these expressions we can now present a total cost model for the diagonal scheme. As before, separate models describe the cases where $p = 2k$ and $p = 2k + 1$. When $p = 2k$, the total cost in seconds per time-step iteration is modeled by

$$\begin{aligned}&\left[\frac{N^3}{2^{2k}}\cdot 485f + 4\left(\frac{N^2}{2^k}\cdot 80b + l\right)\right]\\&\quad + \left(1+\frac{2\,(2^k-1)}{N-2}\right)\left[2\,(N-2)\left(\frac{N}{2^k}\left(\frac{3N}{2^{k+1}}-1\right)\cdot 279f + \frac{N}{2^{k-1}}(40b+l)\right)\right.\\&\qquad\left. + (N-2)\left(\frac{N}{2^k}\right)^2\cdot 953f\right].\end{aligned}$$

For $p = 2k+1$, the total cost in seconds per time-step iteration is modeled by

$$\begin{aligned}&\left[\frac{N^3}{2^{2k+1}}\cdot 485f + 2\left(\frac{N^2}{2^k}\cdot 80b + l\right) + 2\left(\frac{N^2}{2^{k+1}}\cdot 80b + l\right)\right]\\&\quad + \left(1+\frac{3\cdot 2^k-2}{N-2}\right)\left[2(N-2)\left(\left(\frac{3N^2}{2^{2k+2}}-\frac{N}{2^{2k+1}}\right)\cdot 279f + \frac{3N}{2^{k+1}}(40b+l)\right)\right.\\&\qquad\left. + (N-2)\frac{N^2}{2^{\,2k+1}}\cdot 953f\right].\end{aligned}$$

These expressions merit a few words of explanation. We introduce this explanation with brief description of the LU time-step iteration loop. At the top of this loop, the right hand side variables are scaled by the timestep. Then, for each horizontal plane (consisting of the aforementioned tiles) in the computational grid, the left hand side expressions for the lower triangular portion of the LU splitting are calculated (routine jacld) and then relaxed (routine blts). Next for each horizontal plane, left hand side expressions for the upper triangular portion of the LU splitting are calculated (routine jacu) and then relaxed (routine buts). Then, the left hand side variables are updated by adding the latest change in the right hand side. Finally, new right hand side variables are calculated (routine rhs).

We have already detailed the work and communication occurring in the relaxation routines blts and buts and these portions are recognizable in the middle lines of the above two expressions. There are a total of 953 floating point operations per grid point in the jacld and jacu routines combined. This work is reflected in the final line of the previous two expressions. The combined work and communication of the four routines jacld, blts, jacu, and buts is multiplied by the pipeline fill factor which is

$$\left(1+\frac{2\,(2^k-1)}{N-2}\right)$$

for even powers of two processors and

$$\left(1+\frac{3\cdot 2^k-2}{N-2}\right)$$

for odd powers of two processors. Unlike typical pipeline fill costs, this cost occurs not simply once at inception and completion of processing, but is incurred at each timestep. This is because of the symmetric nature of the symmetric successive overrelaxation (SSOR) method, meaning that processing at each timestep proceeds from one corner of the computational grid to the diagonally opposite corner, and then (the "symmetric" portion) in the reverse direction. Recalling that the grid partition consists of vertical pencils through the grid, we see that it is at these stages that processors are initially without work as the pipe fills and again as it flushes. And, we repeat, this occurs at each timestep. The cost of the rhs routine is 485 floating point operations per point. Here we have subsumed the cost of the time scaling of the right hand side variables and the update of the left hand side. The remainder of the top lines in the total cost expressions is the communication cost of the rhs routines. This communication consists of an interchange between processors of adjacent face information. Two layers of face information are exchanged between adjacent processors because the rhs routine calculates a fourth-order dissipation model.

Description of New NPB2.3 LU Scheme

The pencil-based domain decomposition is the same as for the original scheme. As before, each processor works on one tile at a time. Now, however, instead of relaxing points in a diagonal ordering, the standard canonical ordering (column or row ordering) is used. Since the code is implemented in Fortran, we select the column ordering. Relaxation starts at an active corner point ($i = 2$, $j = 2$) of the grid on the bottommost ($k = 2$) active grid plane. The next point to be relaxed has index $i = 3$, $j = 2$, followed by $i = 4$, $j = 2$, and so on, until the end of the first active column ($i = N/p$, $j = 2$) is reached on this first processor. Next, all points on this processor in the next column ($j = 3$) in this ($k = 2$)-tile are relaxed, followed by the ($j = 4$)-column, and so on.

The first communication occurs only after all active points on this processor's ($k = 2$)-tile have been relaxed. The row of values just computed for which i assumes its maximum on this tile are sent to the "eastern" neighbor. Likewise, the column of values for which j assumes its maximum on this tile are sent to the "northern" neighbor. These neighboring processors now begin to relax their points for $k = 2$, and, simultaneously, the first processor proceeds to relax points on its ($k = 3$)-tile. The communication that occurs is of significantly coarser granularity than that of the original scheme. Note also that the ensuing pipeline is fully balanced, which means that once the pipeline is filled (i.e. the pencil in the north-eastern corner of the grid is reached), the load on all processors is completely balanced.

Cost Model for New NPB2.3 LU Scheme

We now estimate the cost for an interior processor tile. As in the above example, we consider the Class A problem on 64 processors. As before, exactly 279 floating point operations are required to relax each point. But now, each interior processor will receive and send a "row edge" of values and a "column edge" of values. An edge of a tile consists of 8 points, each requiring 5 values of 8 bytes each. Therefore, the time required for each interior tile is approximately $64(279f) + 2(8 \cdot 5 \cdot 8b + l)$. Generalizing as before, we find that each tile takes approximately

$$\left(\frac{N}{2^k}\right)^2 \cdot 279f + 2\left(\frac{N}{2^k} \cdot 40b + l\right)$$

seconds to process if $p = 2k$, and

$$\left(\frac{N^2}{2^{2k+1}}\right) \cdot 279f + \frac{3N}{2^{k+1}} \cdot 40b + 2l$$

seconds to process if $p = 2k + 1$, the second term of these expressions representing the communication cost.

Using these expressions we now present a total cost model for the new LU scheme. As before, separate models describe the cases where $p = 2k$ and $p = 2k + 1$. When $p = 2k$, the total cost in seconds per time-step iteration is modeled by

$$\begin{aligned}
&\left[\frac{N^3}{2^{2k}} \cdot 485f + 4\left(\frac{N^2}{2^k} \cdot 80b + l\right)\right] \\
&\qquad + \left(1 + \frac{2\,(2^k - 1)}{N-2}\right)\left[2\,(N-2)\left(\left(\frac{N}{2^k}\right)^2 \cdot 279f + 2\left(\frac{N}{2^k}40b + l\right)\right)\right. \\
&\qquad\qquad \left. + (N-2)\left(\frac{N}{2^k}\right)^2 \cdot 953f\right].
\end{aligned}$$

For $p = 2k + 1$, the total cost in seconds per time-step iteration is modeled by

$$\begin{aligned}
&\left[\frac{N^3}{2^{2k+1}} \cdot 485f + 2\left(\frac{N^2}{2^k} \cdot 80b + l\right) + 2\left(\frac{N^2}{2^{k+1}} \cdot 80b + l\right)\right] \\
&\quad + \left(1 + \frac{3 \cdot 2^k - 2}{N-2}\right)\left[2(N-2)\left(\frac{N^2}{2^{\,2k+1}} \cdot 279f + \frac{N}{2^{\,k}} \cdot 40b + \frac{N}{2^{\,k+1}} \cdot 40b + 2l\right)\right. \\
&\qquad\qquad \left. + (N-2)\frac{N^2}{2^{\,2k+1}} \cdot 953f\right].
\end{aligned}$$

Note that the first and third lines of these expressions are the same as those for the NPB2.0 models, reflecting the fact that only the relaxation section communication scheme has changed.

Observations and Experiments

A glance at the expressions for the generalized time estimates of the relaxation sections of both old and new schemes reveals that the effective computation has been reduced by approximately one third, due to the absence of any intra-pipeline load imbalance. The same reduction holds for the pipeline fill time, since both old and new schemes have to complete work on $2N-1$ tiles successively before all processors are active. The total amount of data communicated is the same in the old and new schemes, but it is in the *number* of communications that the new scheme realizes its major savings. Whereas the original scheme performs $2N/p$ separate communications per tile, the new scheme performs only 2. The savings in time is considerable when communication latency is high, as it is for the majority of modern architectures.

The tables in the Appendix A show comparisons of elapsed times on an IBM SP-Wide Node system for the original (NPB2.0) and new (NPB2.3) LU schemes for the standard class

A, B, and C problem sizes. More information on these benchmark problems, or the source code or accompanying articles, is available on the web site at the URL given in ref.4. We note that the timestep loop of the LU code includes not only the point relaxation and communication sections studied here, but also routines for the formation of the matrix problem. These involve substantial floating point work, as well as a small additional amount of communication. These routines remain unchanged from the original to the new scheme. The small difference in performance of the single-processor computations—which do not involve communication—is due to the difference in the memory reference pattern. (Note that the cost models given are not applicable to the single processor case.) The new scheme, which is column-based, accesses array elements with minimal stride, whereas the old scheme incurs some performance penalties due to its diagonal point access with the associated larger stride. But because at each point a (5×5)-matrix and a (5×1)-vector are accessed, there is still enough locality of memory reference to not degrade the computational performance too much. A larger difference in performance would be observed, for example, for a scalar relaxation problem.

We also note that the smaller problems benefit most from the implementation of the new scheme. That is because the ratio of the communication granularity (defined as number of FLOPs performed for each message sent) of the old scheme over that of the new scheme scales as p/n, implying that for the smaller grids the communication improvement is largest. We make the observation that the communication time of the original scheme constitutes a dominant portion of the total time. This is no longer the case for the new scheme. Appendix C gives communication only costs for the old and new scheme models.

We have generated numbers for the total cost of the old and new schemes. These are given in Appendix B. Comparing these values with those in Appendix A, we see that the cost model is mostly within 30% of the actual timing figures. Values are underpredicted, possibly because the actual runs suffer from real effects such as processor contention by system daemons and network contention. Where the real and predicted values agree, the agreement is likely serendipitous, as such model accuracy is not to be expected. We have also included predictions for performance of two large problem sizes, namely a 256^3 and a 512^3 size. These we have evaluated at up to 16384 processors (a 64 by 64 processor decomposition for the 512^3 size). For this large size problem and this many processors, each processor will still have a realistically sized partition - 8 by 8 by 512 points - from which to construct difference stencils. We see that scaling remains reasonable even to this many processors. In the absence of a machine dependent model for communication network contention, however, these performance numbers must be viewed with some caution.

Finally, we remark that the poor performance of the original scheme was due to an insistence on data-parallel execution of the relaxation loops. When the more general MIMD programming model is adopted, significant savings can be obtained from the coarse-grained pipelining.

References

1. E. Barszcz, R. Fatoohi, V. Venkatakrishnan, S. Weeratunga, *Solution of Regular, Sparse Triangular Linear Systems on Vector and Distributed-Memory Multiprocessors*, NAS Applied Research Branch Report RNR-93-007, NASA Ames Research Center, Moffett

Field, CA 94035, 1993.

2. D. Bailey, T. Harris, W. Saphir, R. Van der Wijngaart, A. Woo, M. Yarrow, *The NAS Parallel Benchmarks 2.0,* NAS Applied Research Branch Report RNR-95-020, NASA Ames Research Center, Moffett Field, CA 94035, 1995.
3. W. Saphir, A. Woo, M. Yarrow, *NAS Parallel Benchmarks 2.1 Results*, NAS Applied Research Branch Report RNR-96-010, NASA Ames Research Center, Moffett Field, CA 94035, 1996.
4. `http://science.nas.nasa.gov/Software/NPB/`

Appendix A: Performance Measurements

LU Class A – IBM SP2 wn			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
1	1923.35	1854.62	1.03
2	1013.83	936.92	1.08
4	558.80	483.44	1.15
8	363.55	255.59	1.42
16	215.84	134.27	1.60
32	141.03	73.67	1.91
64	92.62	42.69	2.16
128	74.69	29.74	2.51

LU Class B – IBM SP2 wn			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
1	8314.27	8121.75	1.02
2	4255.50	4023.66	1.05
4	2326.52	2083.93	1.11
8	1347.51	1042.00	1.29
16	769.83	550.20	1.39
32	487.54	292.58	1.66
64	293.99	155.16	1.89
128	210.10	110.16	1.90

LU Class C – IBM SP2 wn			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
8	4934.32	4163.62	1.18
16	2978.63	2202.81	1.35
32	1898.64	1152.62	1.64
64	1060.69	614.58	1.72
128	670.25	359.57	1.86

Appendix B: Performance Predictions From Models
(For IBM SP2 b=29Mbytes/sec, l=54 microsec)

LU Class A			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
4	695.8	519.6	1.33
8	389.3	269.9	1.44
16	213.9	141.0	1.51
32	130.3	76.8	1.69
64	76.1	42.7	1.78
128	52.1	26.1	1.99
256	32.9	16.9	1.94

LU Class B			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
4	2639.5	2081.7	1.26
8	1422.3	1064.6	1.33
16	758.1	545.8	1.38
32	434.8	286.6	1.51
64	242.8	151.8	1.59
128	153.2	84.9	1.80
256	91.5	49.0	1.86
512	65.4	31.7	2.05
1024	42.5	22.2	1.91

LU Class C			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
4	10131.9	8293.5	1.22
8	5320.7	4203.8	1.26
16	2775.6	2132.9	1.30
32	1522.8	1097.7	1.38
64	821.8	567.0	1.44
128	484.8	302.1	1.60
256	276.0	163.3	1.69
512	181.0	94.6	1.91
1024	110.9	57.3	1.93
2048	82.9	40.1	2.06
4096	55.3	30.7	1.80

LU Size 256x256x256			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
4	38891.4	32625.6	1.19
8	20075.7	16451.0	1.22
16	10320.5	8298.3	1.24
32	5489.0	4222.3	1.29
64	2889.5	2151.6	1.34
128	1619.5	1116.8	1.45
256	887.5	583.1	1.52
512	540.8	317.1	1.70
1024	314.9	176.2	1.78
2048	215.4	107.2	2.00
4096	135.7	69.4	1.95

LU Size 512x512x512			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
4	303612.0	260364.2	1.16
8	154304.2	130719.9	1.18
16	78265.8	65636.2	1.19
32	40418.3	33094.8	1.22
64	20768.8	16693.3	1.24
128	11050.2	8493.1	1.30
256	5814.3	4327.7	1.34
512	3259.9	2245.9	1.45
1024	1785.6	1172.5	1.52
2048	1088.1	637.4	1.70
4096	633.3	354.1	1.78
8192	433.1	215.4	2.01
16384	272.8	139.4	1.95

Appendix C: Total Communication Cost From Model

(For IBM SP2 b=29Mbytes/sec, l=54 microsec)

LU Class A			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
4	121.9	12.2	9.93
8	94.2	10.2	9.16
16	64.6	8.2	7.84
32	51.2	7.4	6.89
64	35.9	6.5	5.48
128	29.7	6.4	4.57
256	21.6	6.3	3.39

LU Class B			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
4	309.7	27.7	11.16
8	236.6	22.4	10.54
16	160.6	17.0	9.40
32	124.8	14.6	8.54
64	86.1	12.0	7.12
128	68.9	11.2	6.15
256	48.8	10.2	4.76
512	41.0	10.4	3.90
1024	30.2	10.6	2.83

LU Class C			
Number of Procs	NPB2.0 (secs)	NPB2.3 (secs)	Ratio
4	781.2	64.6	12.09
8	592.8	50.9	11.62
16	399.8	37.2	10.72
32	306.7	30.6	10.00
64	209.1	23.9	8.74
128	163.7	20.9	7.80
256	113.7	17.9	6.35
512	92.2	17.1	5.38
1024	66.1	16.2	4.07
2048	56.4	17.2	3.28
4096	42.2	18.0	2.34