Model of LPARX multigrid performance on CM-5

A.Zege

May 23, 1995

1 Introduction

The goal of this work was to compare behaviour of a parallel code, in particular its communication pattern, with a simple analytical model and understand how good are predictions of the model. The particular code used for analysis and measurement was a parallel multigrid code from LPARX library [1] written by S.Kohn and S.Baden. LPARX was configured and compiled on CM-5 at Berkeley. The timing and account of communication of the code were implemented with low-level CMMD function calls. Results of the measurements are described and a detailed comparison with a simple model of parallel multigrid, proposed by J.Demmel [2] is given.

It was found that the model gives a very good prediction of the scaling of all measured parameters with the grid dimension. Crude analytical estimation of coefficients in scaling laws is worse — it gives an error of the order of 15-20%. In fact, the model should perform better for larger grid sizes. It was not possble, however, to measure performance for the subgrid (part of the full grid, residing on a single processor) dimensions larger than 512 because of the RAM limitation on the processors.

Unfortunately, given the limitation of the model and the hardware it is not possible to test on Berkeley CM-5 any claim about scaling of measured parameters with the number of processors. There are basically two reasons for that — small number of available partitions and small number of processors in each available partition. Possible numbers of processors, satisfying both the model and the hardware limitations are 4 and 16. For this small number of processors, an average number of neighbours for a given processor depends on the partition size. This was not in fact reflected in the original model, which is assymptotically correct for a large number of processors.

Additinally, hardware parameters best fitting the data were calculated and found in a reasonable accord with the known data.

2 Multigrid performance model

Multigrid is an algorithm for solving Poisson equation [3]; multigrid's principal part is the so-called V-cycle that works in the following stages:

1. coarsening: relaxation is performed on the sequence of grids, where each consequtive grid has twice as less grid points in every dimension than the current grid. It stops at the most coarse grid that may have just several grid points.

2. refining: relaxation is performed on the sequence of grids, where each consequtive grid has twice as much grid points in every dimension as the current grid. For a basic V-cycle refining stops where the coarsening started.

The Fig. 1 shows the basic multigrid cycle along with its recursive pseudocode.

To describe the model of parallel computation, I used parametrization by [2] which defines the time for a processor to send n messages and do m floating point operations as

$$T = T_{calc} + T_{comm}, \quad T_{calc} = mf, \quad T_{comm} = \alpha + n * \beta,$$

where f is time for a floating point operation, α is latency, and inverse of the β is communication bandwidth per processor.

I calculated the time spent by multigrid under the following simplifying assumptions:

- grid is square
- number of processors is a power of 4: $N_{proc} = 2^{2k}$.
- grid dimension of the basic grid dim is a power of 2: $dim = 2^m$. This allows laying out processors in square and simplifies mapping of grid points to processors

These assumptions allow for ideal load balancing between processors on all steps of the multigrid. At a multigrid step where grid has dimension 2^i each processor will own a subgrid with dimension 2^{i-k}. Time t_i spent at the level i of the multigrid is then

$$t_i = N_i^{flops} f + N_i^{mess} \alpha + N_i^{bytes} \beta, \tag{1}$$

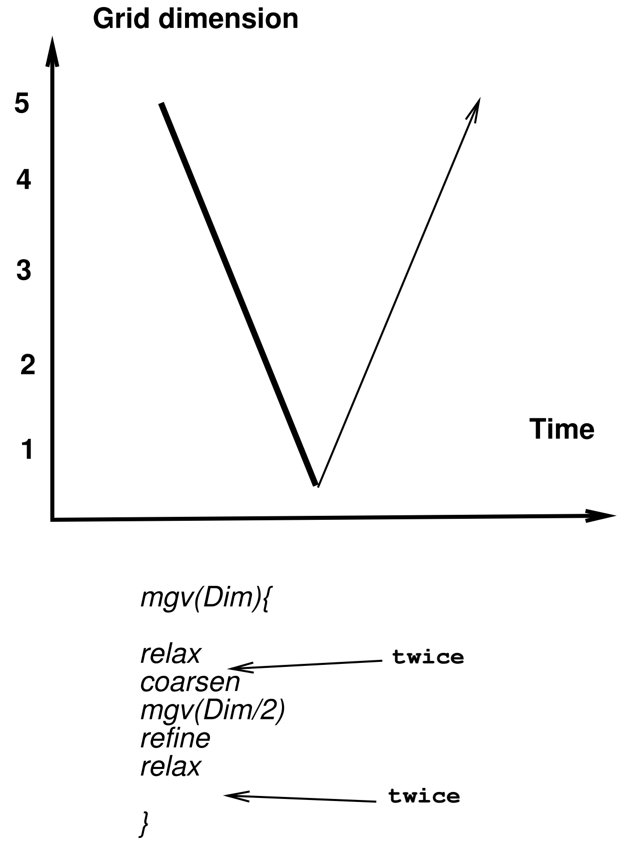

Figure 1: Multigrid V-cycle and multigrid pseudocode. *dim* is dimension of the finest grid

where N_i^{flops}, N_i^{mess}, and N_i^{bytes} are the number of flops, number of messages, and number of bytes send and received. If the number of grid points per processor $L_i = dim_i^2/N_{proc} \gg 1$, then $N_i^{flops} \simeq C_1 L_i, \quad N_i^{mess} = C_2$ and $N_i^{bytes} \simeq C_3\sqrt{L_i}$, where C_1, C_2, and C_3 are constants, are good approximations. The undelying assumptions of those approximations are that the number of floating point operations is proportional to the number of grid points per processor, each processor sends messages only to a constant number of neighbours, and since each message contains just the points on the boundary of the subgrid, the size of the message is proportional to square root of the number of the subgrid points.

Sum of times corresponding to coarsening and refining stages of the multigrid on levels $m \leq i \leq k$ yields

$$T_{k \to m} \approx \frac{4C_1 L_m f}{3} + (log_2 L_m + 1)C_2\alpha + 2C_3 sizeof(gridpoint)\sqrt{L_m}\beta \quad (2)$$

Full time of multigrid is $T = T_{k \to m} + T_{1 \to k}$ One of the ways to deal with coarse grids for $1 \leq i \leq k$ is to continue to split the load and communicate at each step of the V-cycle. Another way is to define minimum grid size. After reaching minimum grid size, processors broadcast their data and continue calculations without further communication. In both cases, cost of calculation and communication depends on the number of processors only (not on the number of total grid points). Since this part of the multigrid is implementation dependent and its contribution to the total multigrid time is not considerable, I chose not to model it. Instead, since this part of time depends only on the number of processors, it is easy to measure it for a small grid and substract from the total time for a larger grids (and the same number of processors). The remainder then would have to correlate with (2).

3 Measurements

The number of messages, bytes transferred, and time spent during the V-cycle of the multigrid LPARX code on CM-5 were measured independently. These measurements were done with CMMD function calls CMMD_node_timer(), CMMD_msg_wait(), and CMMD_mcb_bytes() respectively. Since CM-5 at Berkeley has just 32 processors, there were only two N_{proc} satisfying the assumptions of the model: $N_{proc} = 4$ and $N_{proc} = 16$. The results of the measurements are given in tables 1 and 2

Table 1: $N_{proc} = 4$

Grid dimension	16	64	128	256	512	1024
Number of messages	104	204	256	308	360	412
Number of bytes	832	10168	18352	33192	61344	116120
Time (sec)	0.047	0.152	0.263	0.727	2.426	9.160

Table 2: $N_{proc} = 16$

Grid dimension	64	128	256	512	1024	2048
Number of messages	386	484	582	680	778	876
Number of bytes	29312	38520	52720	76904	121056	205144
Time (sec)	0.205	0.314	0.487	1.005	2.79	9.732

4 Comparison of Model Predictions with Results of Measurements

In order to compare description given by (2) with the measurements of the real code, some specifics of Lparx multigrid implementation should be taken into account.

4.1 Calculation

First, let's count the number of flops on a single multigrid level. Since the cost of refining and coarsening slightly differs, it's more convinient to calculate the total number of flops done for a given grid level during both the coarsening and refining stages. Number of floating point operations in different subroutines of LPARX multigrid code is specified in the table 3, assuming the cost of addition equal to the cost of multiplication. Calculating the total number of flops, I took into account that LPARX code executes two presmoothing and two postsmoothing relaxations [1], so the number of the relaxation flops should be multiplied by four.

4.2 Communication

To figure out communication, let's look on the pseudocode of the parallel functions *refine, coarsen*, and *relax* which are the building blocks of the multigrid V-cycle. All parallel functions have prefix P.

Table 3: Number of Floating Point Operations per grid point in multigrid subroutines (on a single multigrid level).

Subroutine	Number of flops per grid point
relax	14
coarsen	13
refine	9
All	78

Prefine(){
communicate()
synchronize()
interpolate()
syncronize()
}

Pcoarsen(){
restrict()
synchronize()
communicate()
synchronize()
}

Prelax(){
communicate()
synchronize()
relax()
synchronize()
}

Communication happens during execution of *communicate* and *synchronize*.

4.2.1 Number of Messages

I assumed that during execution of *communicate()* processor sends messages to all its neighbours and for syncronization it sends just one message. In the case $N_{proc} = 4$ each processor has three neighbours N^m_{neigh} and in the case $N_{proc} = 16$ choose processor so that $N^m_{neigh} = 5$. Then there are five messages for any of the basic functions in the case $N_{proc} = 4$ and seven messages in the case $N_{proc} = 16$. Model estimation of C_2 is therefore

$$C_2 = 6(N^m_{neigh} + 2),$$

where I took into account that each relaxation executes twice. Comparison between the number of messages on a single level of multigrid calculated from the model and from the best fit to measured data is given in table 4.

Table 4: Number of messages send by a processor on a single multirid level. Comparison between the model and the measurement

N_{proc}	N^m_{neigh}	$N^{messages}$ (model)	$N^{messages}$ (measured)
4	3	30	25.5
16	5	42	49

As table 4 shows, the error of the model estimation for the number of messages is $\simeq 16\%$. Fig.2 and 3 show how well the number of messages for different grid dimensions is approximated by the best fit.

4.2.2 Number of Bytes

Estimating byte transfers, I took into account transfers between processors sharing a boundary and neglected transfers between the processors sharing just a corner. This is a valid approximation because the number of grid points transferred between processors during execution of *communicate()* is proportional to the length of the shared boundary. Analogously, I didn't count bytes transferred during execution of *synchronize()*. Let N^b_{neigh} be the number of neighbours transferring bytes from a given processor. For $N_{proc} = 4$ $N^b_{neigh} = 2$ (see fig. 4 and for $N_{proc} = 16$ I chose the processor such that it has $N^b_{neigh} = 3$.

Analogously to expression for C_2 the formula for C_3 has a form

$$C_3 = 6N^b_{neigh}.$$

Let's compare prediction and measurement for the number of bytes send by a processor, taking into account that grid points are doubles, i.e. $sizeof(gridpointdata) = 8$.

Table 5 gives comparison between the model prediction and the best fit to measured data. As seen from this table, accuracy of prediction is $\simeq 20\%$.

Table 5: Number of bytes sent by a processor per single grid boundary point. Comparison between the model and the measurement

N_{proc}	N^b_{neigh}	N^{bytes} (model)	N^{bytes} (measured)
4	2	192	234
16	3	286	363

Figures 5 and 6 show how well the number of bytes sent is approximated by the best fit.

5 Hardware Parameters

Measurements of N^{mess} and N^{bytes} allowed me to find best fitting values of C_2 and C_3. Using C_1 found from the calculation of the number of flops in the multigrid code and measured values of execution time, I found the best fitting values of parameters f, α, and β. These values are given both for $N_{proc} = 4$ and $N_{proc} = 16$ in the table 6 and compared with the corresponding values for matrix multiplication [2].

Table 6: Hardware parameters, obtained from the best fit to the data

N_{proc}	$f(\mu sec)$	$\alpha(\mu sec)$	$\beta(\mu sec)$
4	0.326	237.2	0.306
16	0.335	383.0	0.065
Matrix multiplication results	0.33	148.5	1.32

6 Comparison Between the Model and the Measurements

Comparing hardware parameters f, α, β, deduced from the measurements of different applications is a one way of checking the validity of results. Another way is to compute purely "theoretical" estimation of the multigrid execution time and compare it with the measured time. In the computation of the "theoretical" estimation I used expressions for the parameters C_i derived earlier and hardware parameters corresponding to matrix multiplication on the CM-5. Comparison between the "theoretical" and the measured values is given in the tables 7, 8 together with the values obtained from the best fit to the coefficients C_i and hardware parameters.

Table 7: Comparison between the multigrid execution time obtained from measurement, best fit and model. $N_{proc} = 4$.

Grid dimension	Measured time	Fitted time	Modelled time
128	0.216	0.229	0.201
256	0.680	0.688	0.651
512	2.379	2.417	2.383
1024	9.113	9.187	9.237

As one can see, the model gives a reliable estimation of the execution time for all values of considered parameters. Sometimes, like in the case of $N_{proc} = 4$ the model gives deceptively good predictions. The reason for this is that the errors for predicted number of messages and number of bytes transferred partially compensate each other.

Table 8: Comparison between the multigrid execution time obtained from measurement, best fit and model. $N_{proc} = 16$.

Grid dimension	Measured time	Fitted time	Modelled time
256	0.282	0.215	0.173
512	0.800	0.686	0.623
1024	2.585	2.446	2.356
2048	9.527	9.354	9.208

7 Conclusion

Figs.2,3,5,6 show that the model gives the right scaling law for dependence of the number of messages and the number of bytes transferred on the grid dimension. Predictions that the model gives for the coefficients in the scaling laws are more crude — the error is of the order of 15-20%. The best fit of data gives a decent estimation of hardware parameters as is shown in the table 6. Surprisingly, multigrid does the same number of Mflops as mutrix multiplication. The values of α and β received from the best fit also seem like having decent accord with the measurements of matrix multiplication (except for the value of β for $N_{proc} = 16$). The values for the execution time obtained from the model have generally a good accord with measured data. For the case of $N_{proc} = 4$ the relative error is less than 7% and for the case of $N_{proc} = 16$ it is less than 20% for three out of four measured grid dimensions.

There are a couple of limitations that make difficult to claim the ability of the model to predict code behavior for the different number of processors. Since the model works only for the number of processors that is the power of four and the total number of processors on Berkeley CM-5 is thirty-two, the only two possible N_{proc} are $N_{proc} = 4$ and $N_{proc} = 16$. Moreover, for these small partition sizes an average number of neighbours for a given processor depends on the partition size. This was not in fact reflected in the original model, which works well for large partition sizes, when the average number of neighbours doesn't depend on partition size.

References

[1] S.Baden and S.Kohn,"LPARX", http: //www-cse.ucsd.edu /users/skohn/lparx.html.

[2] J.Demmel, "Sping 95 CS267 lectures", http: //www.icsi.berkeley.edu/ cs267.

[3] W.Briggs, "A Multigrid tutorial", SIAM, 1987.

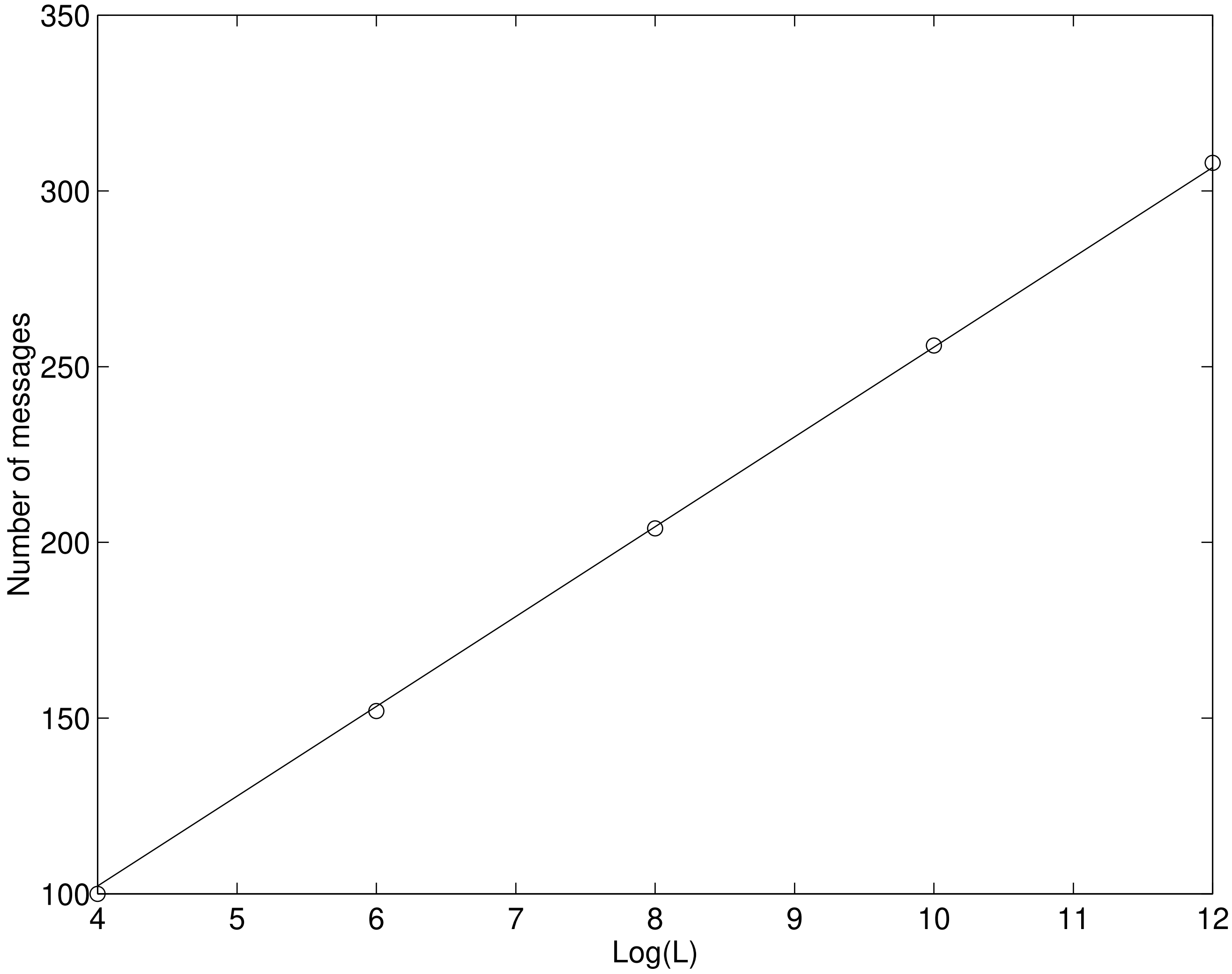

Figure 2: Number of messages sent vs. grid dimension. Circles show measurement, line — best fit. $N_{proc} = 4$

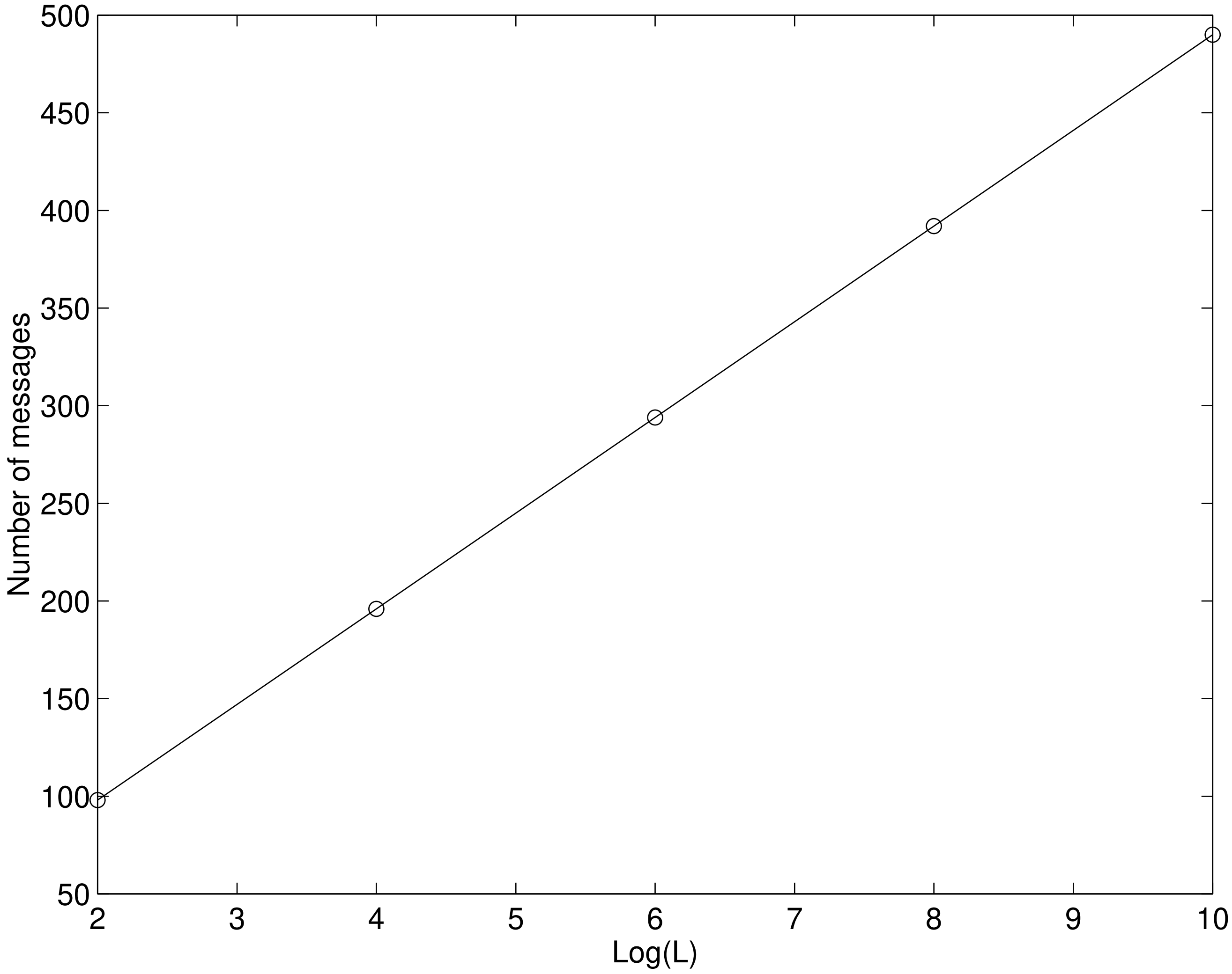

Figure 3: Number of messages sent vs. grid dimension. Circles show measurement, line — best fit. $N_{proc} = 16$

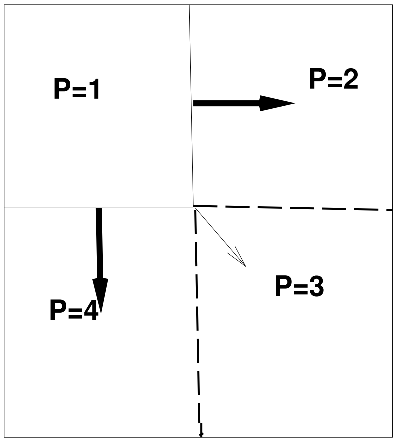

Figure 4: Communication between processors for $N_{proc} = 4$. Byte transfers along direction indicated with a thin arrow are negligible.

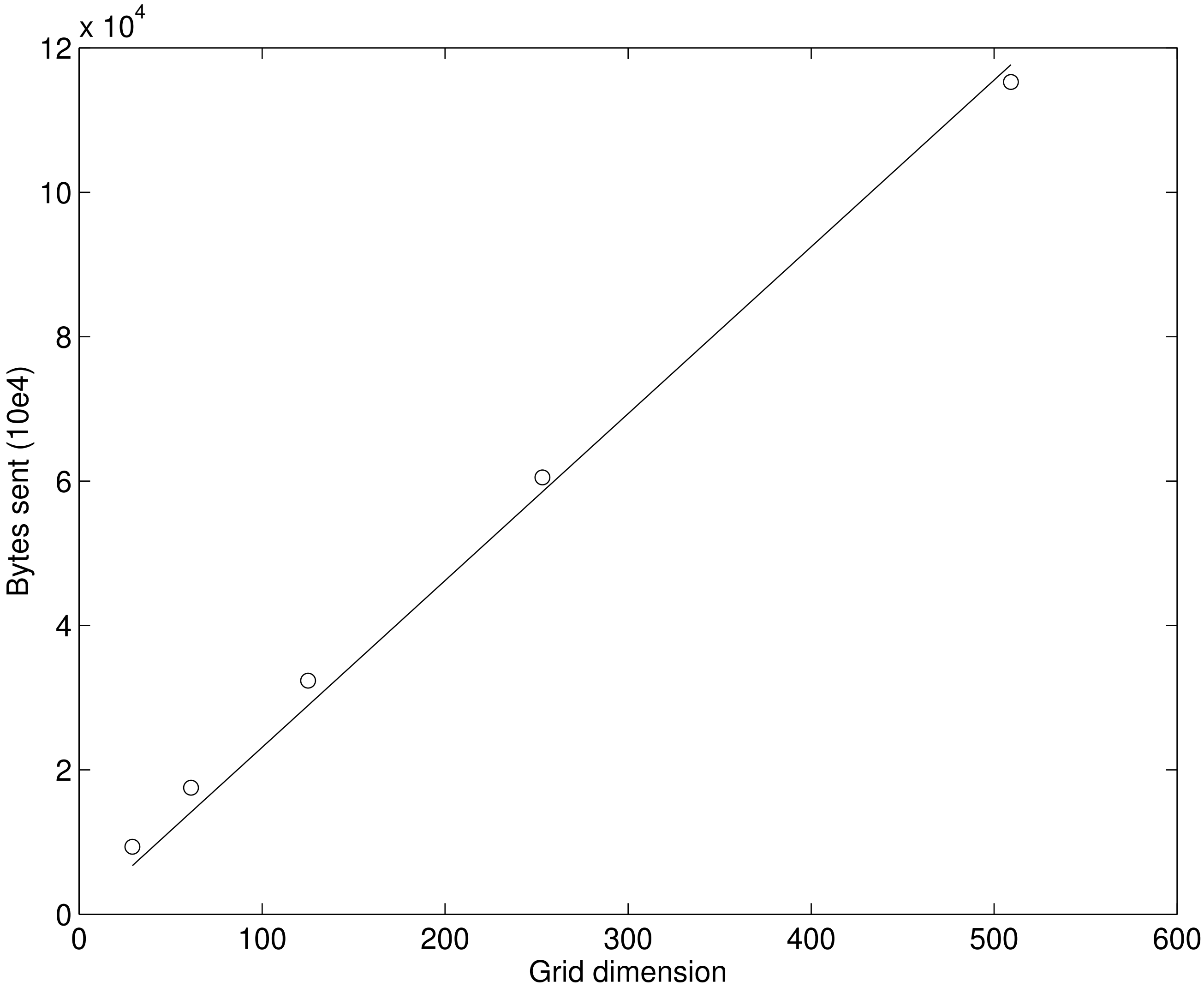

Figure 5: Number of bytes sent vs. grid dimension. Circles show measurement, line — best fit. $N_{proc} = 4$

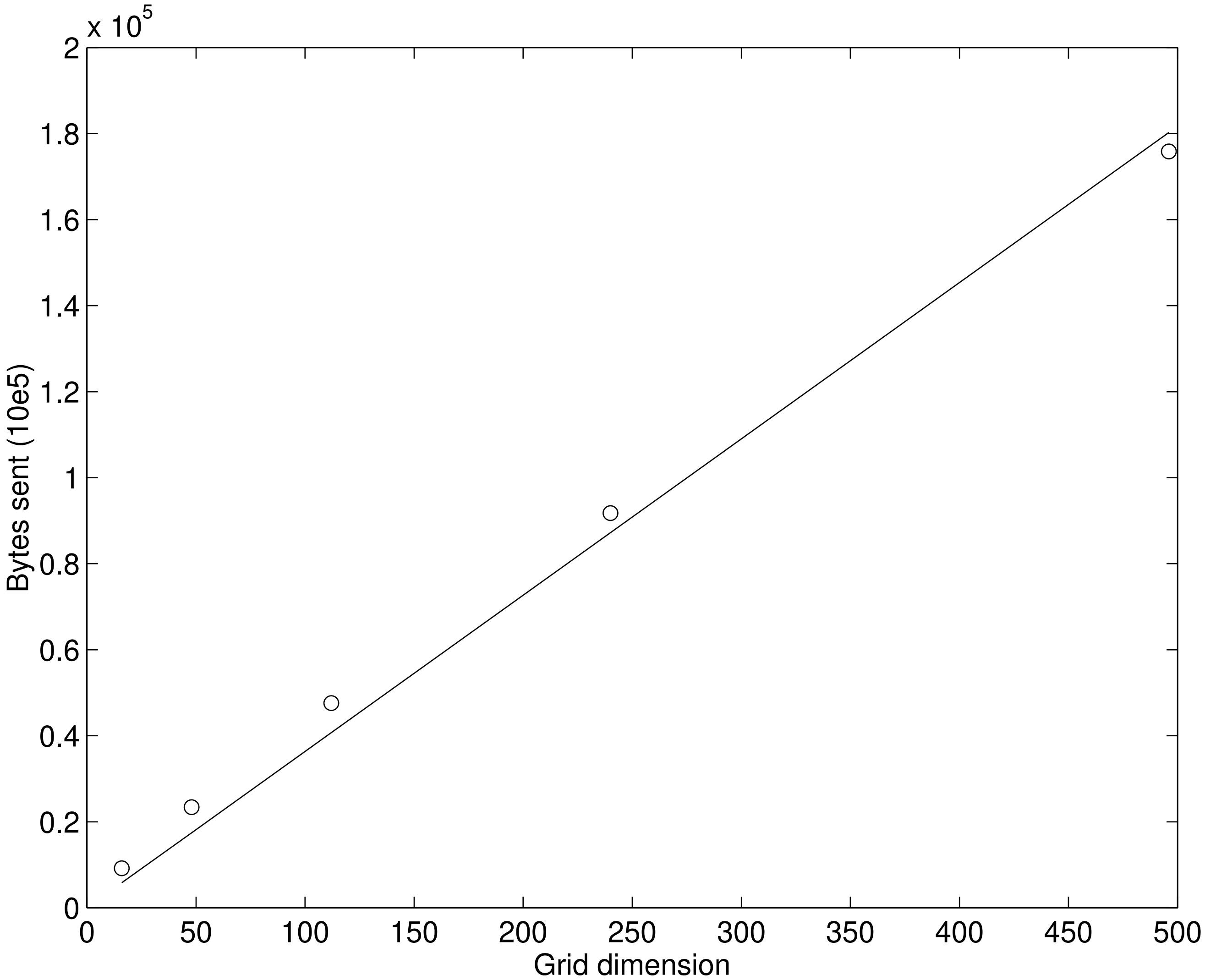

Figure 6: Number of bytes sent vs. grid dimension. Circles show measurement, line — best fit. $N_{proc} = 16$