Introducing Grid Speedup Γ: A Scalability Metric for Parallel Applications on the Grid

Alfons G. Hoekstra and Peter M.A. Sloot

Section Computational Science, Laboratory for Computing,
Systems Architecture and Programming, Faculty of Science,
University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{alfons, sloot}@science.uva.nl
http://www.science.uva.nl/research/scs/

Abstract. We introduce the concept of Grid Speedup as a scalability metric for parallel applications on the grid and analyze it theoretically. It is shown that classes of applications exist for which good grid speedups can be expected. The class of stencil-based applications is taken as an example. The Grid computing community is challenged to demonstrate large grid speedups on real Grid computing environments.

1 Introduction

We consider 'traditional' tightly coupled parallel applications in Grid Computing environments. This is the *Computation-Centric* class of Grid application [1] which, according to Allen et al. "turned to parallel computing to overcome the limitations of a single processor, and many of them will turn to Grid computing to overcome the limitations of a parallel computer." However, common wisdom is that running a tightly coupled parallel application in a computational grid is of no general use because of the large overheads that will be induced by communications between computing elements (see e.g. [2]) and the inherent unreliable nature of a computational Grid.

We introduce the concept of Grid Speedup and analyze it theoretically on a Homogeneous Computational Grid. To do so we assume a two-level hierarchical decomposition. We will show that in terms of Grid Speedup good performance of Computation Centric Grid applications is possible, provided that workloads per Computing Element remain at a sufficiently high level. This large grain size demand is off course not very surprising and well known from parallel computing. Moreover, we introduce grid fractional overhead functions that are sources of grid efficiency reductions. Finally, we will consider one specific example and analyze for which problem sizes grid computing is beneficial.

This new scalability metric will be useful to predict performance of applications on a computational grid, and may be used in e.g. intelligent grid resource managers to facilitate more advanced selection of resources for applications requesting computational power from a Grid.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 245 – 254, 2005.

2 Notation

We use the notation of parallel work W_i as in [3-5]. So, W_i represents the amount of work (expressed e.g. in Mflop) with a degree of parallelism (DOP) i. Define Δ as the computing capacity of a processor (expressed in e.g. Mflop/s). The amount of work executed while running a part of the program with DOP = i is

$$W_i = \Delta i t_i \,, \tag{1}$$

where t_i is the total amount of time during which DOP = i. The total amount of work is

$$W = \sum_{i=1}^{m} W_i \,, \tag{2}$$

with m the maximum DOP in the application. Now assume that the workload W is executed on p processors. The execution time for the portion of work with DOP = i is

$$t_i(p) = \frac{W_i}{i\Delta}\left\lceil \frac{i}{p} \right\rceil . \tag{3}$$

Notice that in the formulation of Eq. [3], possible load imbalance is implicitly taken into account. The total execution time of workload W on p processors, $T_p(W)$, equals

$$T_p(W) = \sum_{i=1}^{m} t_i(p) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}\left\lceil \frac{i}{p} \right\rceil + Q(W,p) \tag{4}$$

with the (communication) overhead for a p processor system for completion of the workload W defined as $Q(W,p)$, and understanding that $Q(W,1) = 0$.

In this paper we assume the simplest possible workload: $W_i = 0$ if $i \neq p$, i.e. a fully parallel workload. In this case we find

$$T_1(W) = T_1(W_p) = \frac{W_p}{\Delta} \,,\; T_p(W) = \frac{W_p}{p\Delta} + Q(W_p, p) \,, \tag{5}$$

and from this we derive expressions for relative speedup S_p and efficiency ε_p

$$S_p = \frac{T_1(W)}{T_p(W)} = \frac{p}{1 + f(W_p, p)} \;;\; \varepsilon_p = \frac{T_1(W)}{pT_p(W)} = \frac{1}{1 + f(W_p, p)} \tag{6}$$

with $f(W,p)$ the fractional overhead function

$$f(W,p) = p\Delta \frac{Q(W,p)}{W} \,. \tag{7}$$

3 Hierarchical Decomposition

We will now consider the case of a parallel application with a workload W running in a grid-computing environment. We assume that within the Virtual Organization (VO)

in which the application is running, the workload is decomposed among C Computing Elements (CE's), and that each CE in itself is some HPC system (typically a parallel computer with p nodes). Such hierarchical resource models were already proposed in [6,7] So, we see a two-level hierarchical decomposition appearing. First, the workload is decomposed between C CE's, and next within each CE the portion of workload is again decomposed between the processors within a specific CE (see Fig. 1). Examples of Grid Enabled parallel applications applying such hierarchical decompositions can be found in the recent literature, e.g. [8].

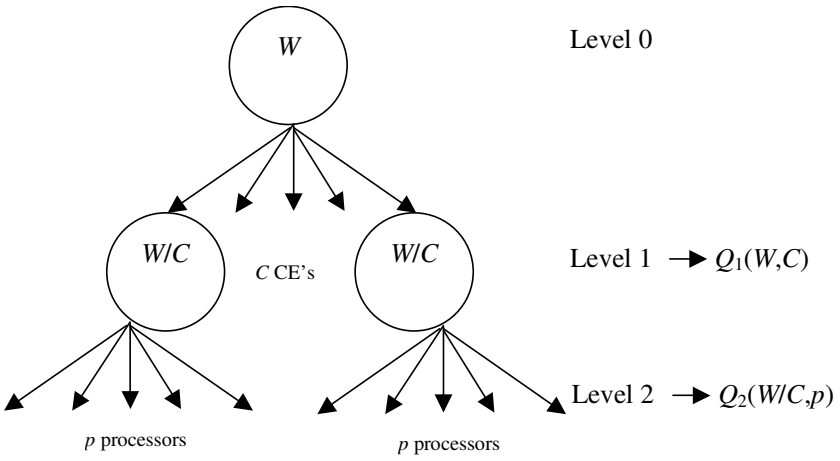

Fig. 1. The hierarchical decomposition

Referring to Fig. 1, we identify three levels. This first is level 0, which is the full non-decomposed workload W. This level would be the sequential computing level. The second level is level-1 decomposition, i.e. the division of the workload between the CE's. This decomposition induces a level-1 overhead $Q_1(W,C)$, e.g. communication between CE's. Next, within each CE the workload W/C is again decomposed between the p processors of the CE. On this level we again encounter an overhead $Q_2(W/C,p)$. If we have one CE (i.e. $C = 1$ and understanding that $Q_1(W,1) = 0$) we return to the standard parallel case where $Q_2(W,p)$ now plays the role of $Q(W,p)$ of the previous section.[1]

Please note that by now we have made the implicit assumption that each CE receives an equal share of the total workload, and that each CE has the same amount of processors. We will work under this assumption. Moreover, we will assume that each processor in each CE has the same computing capacity Δ, and that within each CE the same overhead function $Q_2(W,p)$ applies. We refer to this situation as a *Homogeneous Computational Grid* (HCG). The formalism can be extended to Heterogeneous systems in a straightforward way, using e.g. concepts from [9].

[1] Clearly we can write $Q_2(W,p) = Q(W,p)$.

4 Grid Speedup

We denote the execution time on a HCG with C CE's and p processors per CE as $T_{C,p}(W)$, and with the definitions as introduced earlier we find

$$T_{C,p}(W) = \frac{W_p}{pC\Delta} + Q_2(W_p/C, p) + Q_1(W_p, C) \tag{8}$$

The question now is how to interpret Eq. 8. As already noted in Section 1, common wisdom is that running a application decomposed over several CE's is not very usefull because of the large overheads that will be induced by the communication between the CE's. A first look at Eq. 8 confirms this, the extra overhead expressed by $Q_1(W_p,C)$ contains it all. However, by running the application on more than one CE, we also have more processors to do all the work (a factor C more, reducing the time for pure computation by a factor C). Moreover, the overhead per CE is changed, from $Q_2(W_p,p)$ in the case of running on one CE to $Q_2(W_p/C,p)$ in the case of running on C CE's. Although we do not know this function without turning to specific applications, at this point it is clear that a more detailed analysis is needed.

One can now easily compute the parallel speedup of the application on the HCG. One can however question to what extend this speedup is a good metric to assess the added value of decomposing an application over CE's. Let us recall the reason why parallelism was introduced. Researchers wanted to use parallel computers for two reasons.

1. They were compute bound, meaning that on one processor the computing time was unacceptably high, and more computing power was needed.
2. They were memory bound, meaning that the memory consumption of the application was so large that it would not fit in memory of a single processor. Using more processors (assuming distributed memory computers here) would not only increase the computational power, but also the amount of available memory.

In many cases both reasons applied. To assess the quality of the parallel application, the speedup/efficiency metric was applied. Also, scaled speedup models were introduced, to cope with the fact that researchers will immediately increase the amount of computational work once they get more computing power and available memory.

The bottom line of the previous discussion is that in order to analyze the added value of parallelism, one compared the execution time of the parallel application with a reference value: the execution time on a sequential computer. Let us now analyze performance in a grid-computing environment. We can argue that our reference value in this case should not be the single processor, but the execution time on one single CE. The reason that we decide to decompose our application over more that one CE are exactly the same as our original reasons to parallelize our application, we are compute bound or memory bound in one CE, or a combination of both. So, the question that we must ask ourselves is: "does decomposing over C CE's give us any added value *as compared to* running on one CE?". This leads us to the concept of *Grid Speedup*, defined as

$$\Gamma_p^C = \frac{T_{1,p}(W)}{T_{C,p}(W)}. \tag{9}$$

So, in Grid Speedup we take the quotient of the execution time of our application on 1 CE and the execution time on C CE's. If $p = 1$ we are back in the normal situation of a parallel computation, with C now playing the role of the number of processor. Also note that Grid speedup depends on two parameters, the number of CE's and the number of processors per CE. Grid efficiency is now defined as

$$\gamma_p^C = \frac{T_{1,p}(W)}{CT_{C,p}(W)}. \tag{10}$$

Let us compute the grid speedup for the example of a single parallel workload W_p. Substitute Eq. 8 into Eq. 10, which after some algebra results in

$$\Gamma_p^C = \frac{C}{1 + g_2(W_p, p, C) + g_1(W_p, p, C)}. \tag{11}$$

We defined two fractional grid overhead functions:

$$g_1(W_p, p, C) = C\frac{Q_1(W_p, C)}{T_{1,p}(W_p)}; \tag{12}$$

$$g_2(W_p, p, C) = \frac{CQ_2(W_p / C, p) - Q_2(W_p, p)}{T_{1,p}(W_p)}. \tag{13}$$

The first fractional grid overhead function g_1 plays exactly the same role as the fractional overhead f (Eq. 7) in the simple parallel case. This analogy becomes clear by realizing that f can be rewritten as $f = pQ(W_p,p)/T_1(W_p)$. So, we can obtain good grid speedups if the grid fractional overhead g_1 is small, that is, if we let the grain size, defined as the portion of work per CE, be large enough such that the amount of work per CE is much larger than the amount of overhead induced by the level-1 decomposition Q_1.

The hierarchical decomposition introduces another fractional grid overhead g_2, which expresses the relative difference in overhead inside a CE between the level-1 decomposed workload W_p/p and the original workload W_p. For the special case that

$$CQ_2(W/C, p) = Q_2(W, p) \tag{14}$$

we immediately find that $g_2 = 0$ and

$$\Gamma_p^C = \frac{C}{1 + g_1(W_p, p, C)}. \tag{15}$$

5 An Example: Stencil-Based Operations

Let us consider a prototypical scientific computation: that of a stencil based operation. So, we assume some computational mesh, and we assume an iterative procedure on

this mesh where during each iteration each mesh point is updated, using information from a small local neighborhood of that point. This could e.g. be a time dependent explicit finite difference, finite volume or finite element computation, a Cellular Automaton, or a pixel based image analysis algorithms. In the sequel we assume a two-dimensional square Cartesian mesh with $n \times n$ points.

First consider the case where we apply a 1 dimensional 'strip wise' decomposition of the computational domain. This is schematically drawn in Fig. 2. The left panel shows the decomposition in the normal parallel case (i.e. $C = 1$), and the right panel shows the hierarchical decomposition. Note that in the hierarchical decomposition we choose to decompose the computational domain in each CE along the dimension with the shortest size.

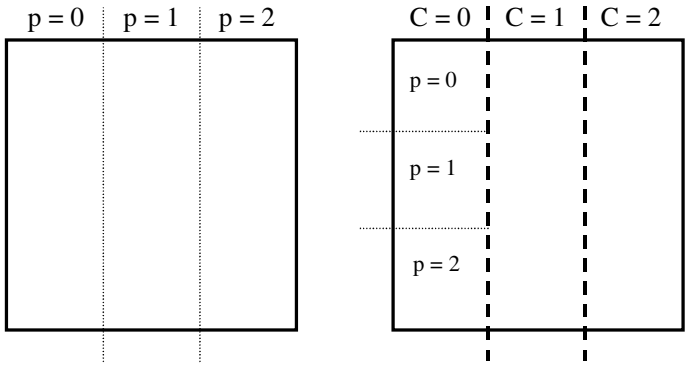

Fig. 2. Decomposition in case 1, strip wise. The left panel shows the decomposition in the normal parallel case, the right panel shows the hierarchical decomposition. In this figure we assume $C = p = 3$

The execution time on one CE in this case becomes

$$T_{1,p} = \frac{n^2}{p\Delta} + 2n\tau_{com}, \tag{16}$$

where τ_{com} is the communication time needed to send the stencil information of a point on the boundary of the processor domain to a neighboring processor. The factor 2 emerges because each processor should communicate with its left and right neighbor.[2] On more than one CE the hierarchical decomposition results in the following execution time:

$$T_{C,p} = \frac{n^2}{pC\Delta} + 2n\tau_{grid} + 2\frac{n}{C}\tau_{com}. \tag{17}$$

Here, τ_{grid} is the communication time needed to send the stencil information of a point on the boundary of a CE to a neighboring CE. Note that we assume here a

[2] For 2 processors this could change from a factor 2 to a factor 1 if the algorithm does not assume period boundary conditions. We neglect this here.

communication pattern where we first communicate between CE's (giving rise to the term $2n\,\tau_{grid}$) followed by a communication step inside each CE (the $2n/C\ \tau_{com}$ term).

Inspection of Eq. 16 and 17 shows that we can write for the overhead functions

$$Q_1(W,C) = 2n\,\tau_{grid} \tag{18}$$

$$Q_2(W/C,p) = 2\frac{n}{C}\tau_{comm} \tag{19}$$

In this example $CQ_2(W/C,p) = Q_2(W,p)$ applies, so we find immediately for the Grid speedup, using Eq. 15,

$$\Gamma_p^C = \frac{C}{1+\dfrac{2Cn\,\tau_{grid}}{\dfrac{n^2}{p\Delta}+2n\,\tau_{com}}}, \tag{20}$$

which we rewrite to

$$\Gamma_p^C = \frac{C}{1+\dfrac{2C\alpha}{\beta+2}}, \tag{21}$$

with

$$\alpha = \frac{\tau_{grid}}{\tau_{com}}, \tag{22}$$

$$\beta = \frac{n}{p\Delta\tau_{com}}. \tag{23}$$

Note that dimensionless parameter α expresses the imbalance between inter- and intra CE communication. The dimensionless number β contains the grain size of the application running on 1 CE, and the balance between computational speed and communication within one CE. Note that if one would compute the fractional communication overhead in this example (Eq. 7) one would find that $f = 2/\beta$. Large grid speedups are obtained if α is very small and/or β is very large.

We should explicitly analyze the case of $C = 2$ and assuming no periodic boundary conditions in our application. In that case we have

$$Q_1(W,C=2) = n\,\tau_{grid} \tag{24}$$

and Eq. 21 changes to

$$\Gamma_p^{C=2} = \frac{2}{1+\dfrac{2\alpha}{\beta+2}}. \tag{25}$$

The first question one could ask is: "when does it pay off to execute my application on more than one CE?". Translated in our formalism, when will $\Gamma_p^C \geq 1$. Clearly, a

speedup marginally larger then 1 is probably not worth the effort of porting an application to the grid. A better analysis would be to demand that the Grid efficiency should be larger than a certain value, say larger then γ_0. So, we demand that $\Gamma_p^C / C \geq \gamma_0$. This results in

$$\beta \geq \begin{cases} \dfrac{2\gamma_0\alpha}{1-\gamma_0} - 2 \text{ if } C = 2 \textit{ and} \text{ no periodic boundaries apply} \\ \dfrac{2\gamma_0\alpha C}{1-\gamma_0} - 2 \text{ otherwise} \end{cases} \tag{26}$$

Note that if we put $\gamma_0 = 1/C$ we return to demanding that the grid speedup should be larger than 1.

Let us now try to find some representative numbers for the parameters, and compute grain sizes for which we expect to see a beneficial effect of using a Grid for tightly coupled applications.

As a representative application that falls in the category of the stencil based operations as analyzed in this section we choose the Lattice Boltzmann Method [10]. As was recently reported by Wellein et al. during ICMMES 2004 in Braunschweig [11] a Lattice Boltzmann simulation attains computational speeds in the order of 1 Gflop/s on modern processors such as a 1.3 GHz Itanium2 or a 2.4 GHz Opteron. Updating one lattice point requires in the order of 200 floating point operations, so we find $\Delta = 1\times10^9 / 2\times10^2 = 5\times10^6$ sites/s. Now let us assume that $\alpha = 10$, i.e. inter CE communication is 10 times faster then intra CE communication (see e.g. discussion in [12]). Finally, assume that processors in a CE can communicate at a speed of 10 Mbyte/s and that in a 2D Lattice Boltzmann simulation 4 floating point numbers must be sent in both directions. Under these assumption we find $\tau_{\text{comm}} = 4\times8 / 10^7 \sim 3\times10^{-6}$.

Let us first consider the case of two CE's. In this case the result is that (see Eq. 26, using $\alpha = 10$)

$$\beta \geq \frac{20\gamma_0}{1-\gamma_0} - 2 . \tag{27}$$

The breakeven point is for $\gamma_0 = 1/C = 0.5$, so $\beta \geq 18$. Using Eq. 23 we find

$$\frac{n}{p} \geq 18\Delta\tau_{comm} . \tag{28}$$

With the numbers as defined above Eq. 28 results in $n/p \geq 270$. So, only if $n \geq 270\times p$ the Grid Speedup will be larger than 1 and therefore the execution time of running the simulation on 2 CE's will be smaller than running on 1 CE. Now suppose we want to have a grid efficiency of at least 0.8 (i.e. a grid speedup of 1.6 on 2 CE's). In that case we find $n/p \geq 1170$. We clearly see that in order to get real benefits of running this stencil-based operation on a Computational grid we need very large domains. Assuming e.g $p = 10$ (which would be a relatively small parallel computer) boils down to demanding that in order to pay off, $n \geq 2700$, and in order to get real

benefit $n \geq 11700$. The positive conclusion is that albeit large, these are not unrealistically large grain sizes and that such applications can truly achieve grid speedup on 2 CE's. Clearly for $C > 2$ larger dimensions are needed, and our analysis gives a quick estimate of what to expect.

6 Discussion and Conclusions

We have introduced a new metric, Grid Speedup Γ, that allows analyzing the performance of tightly coupled parallel applications on a Homogeneous Computational Grid. Using the concept of a two level hierarchical decomposition of the workload, a general formalism was introduced that allows computing Grid Speedup in terms of two fractional overhead functions. Using this formalism we analyzed in detail a prototypical application based on a two-dimensional stencil operation. In turns out that two dimensionless numbers now determine Grid Speedup. The number α expresses the ratio of inter- and intra CE communication and the other, β, is inversely proportional to the well-known fractional communication overhead of the parallel application. Estimating the parameters appearing in the model leads to the conclusion that for a small number of CE's good Grid Speedups is attainable, as long as the work per CE is large enough.

More extended analysis then the one presented here clearly is possible. For instance, in our example a more efficient decomposition, in which the intra- and inter CE decomposition run in the same direction, can be used, leading to the count intuitive conclusion that for such decomposition the Grid Speedup can be larger due to a latency hiding effect. A more realistic case in which CE's have different number of processors, leading to the concept of a fractional C, could be discussed. Or, more realistic workloads, including a sequential workload W_1 could be considered. We will analyze all this in more detail in a future paper.

We must now test these theoretical ideas and try to measure Grid Speedups on real Computational Grids. It will be quite challenging to try to apply our theoretical ideas to real Grids, that will behave must more dynamic and erratic as our ideal Homogeneous Computational Grid. Fortunately test systems, such as the Dutch ASCI – DAS2 computer [13], that come close to the idea of a Homogenous Computational Grid are available as a first test site, before embarking on scalability measurement on real Grids, such as the European CrossGrid test bed [14].

References

1. Allen, G., Goodale, T., Russell, M., Seidel, E., Shalf, J.: Classifying and Enabling Grid Applications. In Berman, F., Fox, G.C., Hey, A.J.G. (Eds.); Grid Computing, Making the Global Infrastructure a Reality. Wiley, chapter 23 (2003)
2. Lee, C., Talia, D.: Grid Programming Models: Current Tools, Issues and Directions. In Berman, F., Fox, G.C., Hey, A.J.G. (Eds.): Grid Computing, Making the Global Infrastructure a Reality. Wiley, chapter 21, specifically section 21.2.3 (2003)
3. Hwang, K.: Advanced Computer Architecture, Parallelism, Scalability, Programmability. McGraw-Hill, New York, chapter 3 (1993)

4. Kumar, V., Gupta, A.: Analyzing Scalability of Parallel Algoritms and Architectures. J. Parallel Distrib. Computing 22 (1994) 379-391
5. Sun, X.H., Ni, L.M.: Scalable Problems and Memory-Bounded Speedup. J. Par. Distr. Comp. 19 (1993) 27-37
6. Halderen, A.W. van, Overeinder, B.J. Sloot, P.M.A., Dantzig, R. van, Epema, D.H.J, Livny, M.: Hierarchical Resource Management in the Polder Metacomputing Initiative. Parallel Computing 24 (1998) 1807-1825
7. Iskra, K.A., Belleman, R.G., Albada, G.D. van, Santoso, J., Sloot, P.M.A., Bal, H.E., Spoelder, H.J.W., Bubak, M.: The Polder Computing Environment, a system for interactive distributed simulation. Concurrency and Computation: Practice and Experience 14 (2002) 1313-1335
8. Keller, R., Gabriel, E., Krammer, B., Müller, M., Resch, M.M: Towards Efficient Execution of MPI Applications on the Grid: Porting and Optimization Issues. J. Grid Comp. 1 (2003) 133-149
9. Bosque, J.L., Pastor, L: Theoretical analysis of scalability on heterogeneous clusters. In proceedings of the 4th IEEE/ACM International Conference on Cluster Computing and Grids, Chicago (published on CD), April 2004.
10. Succi, S: The Lattce Boltzmann Equation for fluid dynamics and beyond. Oxford Science Publications (2001)
11. Wellein, G. et all.: Optimization Approaches and Peformance Characteristics of Lattice Boltzmann Kernels. Presented during International Conference for Mesoscopic Methods in Engineering and Science, Braunschweig, July 2004. To be published in Computers and Fluids.
12. Berman, F., Fox, G., Hey, T.: The Grid: past present future. in Berman, F., Fox, G.C., Hey, A.J.G. (Eds.): Grid Computing, Making the Global Infrastructure a Reality. Wiley, chapter 1 (2003)
13. Bal, H.E. et all.: The Distributed ASCI supercomputer project: Operating Systems Review 34 (2000) 76-96
14. Gomes, J. et all.: First prototype of the CrossGrid Testbed. in Rivera, F.F., Bubak, M., Tato, A.G., Doallo, R. (Eds.): Grid Computing, Lecture Notes in Computer Science, Vol. 2970, . Springer-Verlag, Berlin Heidelberg New York (2004) 67-77