Dynamic load balancing algorithm for continuum mechanics problems with essential redistribution of workloads among the processes

Artem Karpeev
Russian Federal Nuclear Center – Zababakhin All-Russian Research Institute of Technical Physics, Snezhinsk, Russia
E-mail: a.v.karpeev@vniitf.ru

Abstract. A number of important applications have computational requirements that vary over time in an unpredictable way. For the efficient usage of distributed-memory parallel systems when solving such problems a new dynamic load balancing algorithm is suggested and explored. It allows to raise the load efficiency for model task up to 70% with balancing frequency $\nu_{\text{bal}} \approx 0.1$.

1. Introduction
The efficient use of a parallel computer requires two, often competing, objectives to be achieved: the processor must be kept busy doing useful work; the amount of interprocessor communication must be kept small. For many problems in scientific computing, these objectives can be obtained by a single assignment of tasks to processors that do not change throughout the simulation. Calculations amenable to such a static distribution include traditional finite element and finite difference methods, dense linear solvers, and iterative solvers. However, some important applications have computational requirements that unpredictably vary over time. These kinds of computations seem to be particularly prevalent in computational mechanics and include the following applications: adaptive mesh refinement, adaptive physics models, particle simulations, multiphysics simulations. For such applications, high performance can only be obtained if the workload is distributed among the processors in a time-varying, dynamic fashion.

1.1. Definitions
Defining the method for estimation of computational efficiency, we are using general principles of parallel computing in mechanics. Computing is assumed to be done by time steps. They can be divided into two parts: calculation in main subroutine ($t_{\text{calc}}$) and exchange data by adjacent parallel processes ($t_{\text{exch}}$). All parallel processes are synchronized each time step, so the processes that do calculation faster, have to wait for the slowest process at the synchronization point. If $\tau_{p,n}^{\text{exch}}$ – an astronomical time when the parallel process comes to the exchange point, then it waiting time at the time step is defined as

$$
t_{\text{wait}}^{p,n} = \max_p \{ \tau_{p}^{\text{exch}} - \tau_{p}^{n,\text{exch}} \}. \tag{1}
$$
As a result, the step time should be defined as a sum of calculation, exchange, and waiting times
\[ t_{\text{step}}^{p,n} = t_{\text{calc}}^{p,n} + t_{\text{exch}}^{p,n} + t_{\text{wait}}^{p,n}. \]  
(2)

The workload efficiency for any parallel process at the time step is defined as a ratio of the useful work time \( t_{\text{calc}} + t_{\text{exch}} \) to the total step time \( K_{\text{load}}^{p,n} = \left( 1 - \frac{t_{\text{wait}}}{t_{\text{step}}^{p,n}} \right) \cdot 100\% \), \( p = 1, \ldots, N_p \),

(3)

where \( N_p \) – the number of parallel processes. The average load efficiency of the time step is determined as

\[ K_{\text{load}}^n = \frac{\sum_{p=1}^{N_p} K_{\text{load}}^{p,n}}{N_p}. \]  
(4)

If the exchange time is much less than the calculation time, the average load efficiency can be evaluated by knowing only the spatial distribution of computational workload

\[ t_{\text{wait}}^{p,n} = \max_{p \in \{1, \ldots, N_p\}} \left\{ t_{\text{calc}}^{p,n} \right\} - t_{\text{calc}}^{p,n}, \quad K_{\text{load}}^n \approx 100\% \cdot \frac{\sum_{p=1}^{N_p} t_{\text{calc}}^{p,n} / N_p}{\max_{p \in \{1, \ldots, N_p\}} \left\{ t_{\text{calc}}^{p,n} \right\}}. \]

1.2. Test problem

As a test problem, we are considering a task called "implosion of shells with increased workload". It mimics spherical symmetric workflow in a simulation of the ICF problem [1, 2] on an Eulerian mesh. Regions with increased workload are spherical shells, whose positions in space and time are defined by the function \( C(r,t) \):

\[ C(r,t) = \begin{cases} 
K_1, & \text{if } |r| \in [R_1(t), R_1(t) + D_1], \\
K_2, & \text{if } |r| \in [R_2(t), R_2(t) + D_2], \\
1, & \text{otherwise};
\end{cases} \]  
(5)

where \( K_i \) – workload increase factor, \( D_i \) – thickness of the \( i \)th shell with the increased workload, \( R_i(t) \) – a time dependence of the \( i \)th inner shell radius. In general, the shell movement is nonlinear and \( R_i(t) \) should be defined by the expression

\[ R_i(t) = R_{i,0} - \int_{t_0}^{t} V_i(t) \, dt, \]  
(6)

where \( R_{i,0} \) – initial radius, \( V_i(t) \) – radial velocity. But we are interested in the explicit computational algorithm where the time step is taken in such a manner that the characteristics movement through the mesh is linearized. So, it is enough to study a constant radial shell velocity case

\[ V_i(t) = V_{i,0} > 0, \quad R_i(t) = R_{i,0} - V_{i,0} (t - t_0). \]  
(7)
1.3. Regular decomposition without balancing
Before exploring different load balancing algorithms, it is worth evaluating the workload efficiency for our test problem in a calculation with static regular domain decomposition – figure 1. The curves color defines the number of parallel processes. It can be seen that the efficiency decreases as the number of parallel processes increases. Most of the curves lie below 0.5, indicating that half of the computing resources are not used and their increase does not allow reducing the calculation time.

2. Load balancing model
The ultimate goal of any load balancing algorithm, static or dynamic, is to improve the performance of a parallel application. To do that, the balancer defines how the objects comprising a computation are distributed between the processes. Objects can be any entity that the application treats as indivisible. Since the objects are assumed to have variable workloads, we need instruments for their estimation. It seems natural to use run-time measurements. For most applications in computational mechanics, the mesh elements or particles become natural objects. But it is not possible to profile each cell, so usually, the full region is profiled and then some empirical estimation for the spatial workload distribution is used.

2.1. Load balancing objects
We propose a different way for a definition of the spatial workload distribution and introduce a new level of decomposition – boxes. They are defined by splitting the initial uniform domain decomposition into \( N_s = 2^{d \alpha} \) equal parts, where \( d \) – the space dimension, \( \alpha \in \{0, 1, 2, \ldots\} \) – the discretization factor which is the ratio of boxes to the number of parallel processes, \( \alpha = 0 \) corresponds to the regular decomposition.

The balancing goal is to redistribute the boxes between the parallel processes so that the total workload of the boxes assigned to each process should be as close to its average value as possible.

If the set of boxes is defined as

\[
B = \{b_i : i = 1, 2, \ldots, N_p \cdot N_s = N_b\},
\]
then the goal of balancing is to define the orthogonal subsets of boxes

\[
\{B^p : p = 1, \ldots, N_p\} / \quad B^l \bigcap B^m = \emptyset; \quad \bigcup_{p=1}^{N_p} B^p = B; \quad \max_{p \in \{1, \ldots, N_p\}} \{W^{p,n}\} \geq K_{th},
\]

where \(W^{p,n}\) – the \(p^{th}\) parallel process computational workload on the \(n^{th}\) time step, that is defined as

\[
W^{p,n} = \sum_{j \in B^p} w^n_j,
\]

\(w^n_j\) – the measured box workloads, \(\bar{w}^n\) – the average workload that defined by the expression

\[
\bar{w}^n = \frac{\sum_{p=1}^{N_p} W^{p,n}}{N_p} = \frac{N_b}{N_p} \sum_{j=1}^{N_b} w^n_j,
\]

\(K_{th} \in (0, 1)\) – the given efficiency threshold.

2.2. Kernighan-Lin algorithm

If we consider a case where the time of exchange much less than the time of calculation for any distribution of boxes over the parallel processes, then the Kernighan-Lin (K-L in abbreviation form) algorithm of graph splitting [3] is optimal. It begins with sorting workloads of boxes in decreasing order and defining per-process empty subsets. Then we split the sorted list of boxes using the greedy algorithm. After that the obtained decomposition is iterated of K-L swaps reducing the deviation of the most loaded process.

Figure 1 shows K-L algorithm results working on the test problem. Particularly, plots show the balancing frequency \(\nu_{bal}\) versus the number of parallel processes for different values of efficiency threshold and two values of the discretization factor: \(\alpha = 1\) (on the left side) and \(\alpha = 2\) (on the right side). This measure shows how often used balancer. As we can see the balancing frequency increases both as the efficiency threshold increases and as the number of parallel processes increases. For \(\alpha = 1\) and \(N_p > 64\) balancing is done each time step but \(K_{load} \leq 0.8\). Values above the bars define a measure of imbalance in the distribution of data volume between the processes, i.e.

\[
\Delta N = \frac{n_{,p \in \{1, \ldots, N_p\}} N (B^{p,n})}{N_s},
\]
Figure 2. Balancing frequency vs the number of parallel processes for different $K_{th}$: left – $\alpha = 1$, right – $\alpha = 2$.

$N(B_{p,n})$ – count of boxes on a process. One can see that when $\alpha = 1$, effective computing with $N_p > 16$ requires almost twice as much memory. It can be too expensive when the resources are limited. Increased discretization factor allows reducing the balancing frequency and imbalance factor.

2.3. Algorithm with Space-Filling Curve

If we want to keep locality, one of the most widely used methods for fast domain decomposition is based on the use of Space-Filling Curves (SFC in abbreviation form) [4]. In this work we use the Z-curve. An example of box ordering along the Z-curve is shown on the figure 3. Knowing the average workload, the decomposition of boxes ordered along the SFC is trivial. The initial point coincides with the beginning of the SFC. Each next splitting point is defined by the position where boxes workload integral along the SFC is getting higher than the average value. Advantages of this algorithm – it is fast and simple. If we use the SFC algorithm in calculation with spatial workload distribution close to uniform, it gives box distribution preserving the communication pattern of a regular partition.

\begin{verbatim}
1: Function SplitBoxesSFC({$w^n_i$})
2: EmptyDist() $\Rightarrow$ {W_{p,n}}
3: {$w^n_i$} $\Rightarrow$ sortSFC() $\Rightarrow$ {sw^n_i}
4: p = 0
5: for (i=0; i < N_b; i++) do
6: if ((W_{p,n} + sw^n_i) > $\bar{w}$) then
7: p = p + 1
8: end if
9: sw^n_i $\rightarrow$ W_{p,n}
10: end for
11: end Function
\end{verbatim}

Figure 3. Box ordering along the Z-curve.

Figure 4. Decomposition algorithm with SFC ordering.
2.4. Hybrid algorithm
Both of the considered above load balance algorithms have their merits and demerits. Trying to combine their merits and reduce the influence of demerits we formulated next hybrid algorithm. First of all, we define a uniform distribution of boxes with the SFC ordering. It is used as an initial approximation for each call of the decomposition procedure. If the efficiency of the derived decomposition higher than the given threshold, it is accepted and the algorithm returns the result. So, we get fast effective decomposition for the problem with workload distribution close to uniform. If the efficiency of decomposition is lower than the given threshold, it is used as an initial approximation for the K-L algorithm with a reduced amount of swaps. The hybrid algorithm allows saving all advantages of the parents algorithms. Disadvantage – sometimes it cant get results with given efficiency. If we are comparing balancing frequency versus the number of MPI processes for \( K_{th} = 0.7 \), \( \alpha = 2 \). The left figure – 2D case, the right one – 3D. Color defines the algorithm: green – Kernighan-Lin, blue – hybrid, red – SFC. As we can see, the hybrid algorithm allows reducing balancing frequency and imbalance in the distribution of data volume between the processes in comparisons with SFC. At the same time, we can control the imbalance in the distribution of data volume between the MPI processes.

### Algorithm 2 Decomposition with hybrid scheme

```plaintext
1: Function SplitBoxesHybrid(\( \{w^n_i\} \))
2: \( \{w^n_i\} \) ⇒ SplitBoxesSFC\( (N_{\text{max}}^b) \) ⇒ \( \{W^p,n\} \)
3: if \((\text{getEfficiency}(\{W^p,n\}) > K_{th})\) then
4: return \( \{W^p,n\} \)
5: else
6: return SwapKL(\( \{W^p,n\} \), \( N_{\text{max}}^b \))
7: end if
8: end Function
```

3. Conclusion
We have studied two widely used approaches for dynamic load balancing. A hybrid algorithm based on them has been created. It allows increasing the load efficiency on the test problem...
to the value higher 70% with a frequency of balancing approximately 0.1. The algorithm has several useful properties.

(i) If the algorithm is applied to problems with a weak deviation of the workload distribution from the uniform one, it preserving the communication pattern of a regular decomposition;

(ii) At any redistribution of boxes between the processes, their quantity on each one remains limited.

(iii) If we use appropriate partition options, the localization of adjacent blocks increases. This reduces the volume of interprocessor messages or network load.

References
[1] Youngs D L and Williams R J R 2008 Int. J. Numer. Meth. Fluids 56 1597–1603
[2] Mongomery D S and et al 2018 Physics of Plasmas 25
[3] Kernighan B and Lin S 1970 Bell System Technical J. 29 291–307
[4] Sagan H 1994 Space-Filling Curves (Springer)