CSRD (2009) 23: 225–230
DOI 10.1007/s00450-009-0071-y

SPECIAL ISSUE PAPER

Architecture aware parallelization of solvers for PDE systems on geometrical graphs

Sergiy Y. Gogolenko · Volodymyr Svjatnyj

Published online: 30 April 2009

Abstract Solving PDEs on geometrical graphs with method of lines approach leads to large-scale, homogeneous, weakly connected ODE systems. Such differential equation systems can be efficiently solved on parallel computers by exploiting of parallelism across system. In this case optimal parallelization of the ODE solvers is equivalent to finding an optimal mapping of secondary topology graph on architecture graph. Architecture aware graph partitioning is a relatively new direction of research. Available solutions do not cover all the most important hardware platforms. Furthermore, usage of existing architecture aware partitioners does not provide facilities for estimating discretization parameters in PDE solvers. In this paper, we discuss an approach to overcome above-mentioned drawbacks.

Keywords Dynamic network object · Partial differential equation on geometrical graph · Method of lines · Architecture aware partitioning · Statistical optimization methods · Bayesian heuristic approach

S. Y. Gogolenko (✉)
Max Planck Institute for Dynamics of Complex Technical Systems,
Sandtorstr. 1,
39106 Magdeburg, Germany
e-mail: gogolenk@mpi-magdeburg.mpg.de

V. Svjatnyj
Computer Science Department, Donetsk National Technical University,
Artem str. 57,
83000 Donetsk, Ukraine
e-mail: svjatnyj@cs.dgtu.donetsk.ua

1 Introduction

Majority of models for dynamic network objects (DNOs) can be represented as nonlinear PDE systems on geometrical graphs

$$\boldsymbol{F}\left(t, \boldsymbol{u}, \frac{\partial \boldsymbol{u}}{\partial t}, \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{\Gamma}}, \frac{\partial^2 \boldsymbol{u}}{\partial t^2}, \frac{\partial^2 \boldsymbol{u}}{\partial \boldsymbol{\Gamma}^2}\right) = 0\,, \tag{1}$$

where geometrical graph $\boldsymbol{\Gamma} = (V_{\boldsymbol{\Gamma}}, E_{\boldsymbol{\Gamma}})$ corresponds to the graph of DNO [1]. Equations on the edges $e \in E_{\boldsymbol{\Gamma}}$ reflect behavior of the DNO along the edges of $\boldsymbol{\Gamma}$. Equations in the vertices $v \in V_{\boldsymbol{\Gamma}}$ define laws of interaction between parts of the DNO (transmission conditions).

Prevalent method for simulation of these models is half discretization of PDEs using the method of lines (MOL) approach. It leads to large-scale homogeneous weakly connected ODE or DAE systems on non-weighted secondary topology graphs. If half discretization in space (with a space step ε) is used, the secondary topology graph $\boldsymbol{G}_{\boldsymbol{\varepsilon}} = \boldsymbol{G}(\varepsilon, \boldsymbol{\Gamma})$ is obtained from $\boldsymbol{\Gamma}$ by replacing each of its edges by a path of length $[l_i/\varepsilon]$, where l_i is a weight of edge $e_i \in E_{\boldsymbol{\Gamma}}$, or, equivalently, by inserting of additional $[l_i/\varepsilon] - 1$ vertices into each edge of $\boldsymbol{\Gamma}$, and ODE system takes the form

$$\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}t} = \boldsymbol{F}_{\boldsymbol{G}_{\boldsymbol{\varepsilon}}}(t, \boldsymbol{y})\,, \tag{2}$$

where $\boldsymbol{y} : \mathbb{R}^+ \to \mathbb{R}^m$ denotes vector of state variables on the secondary topology graph $\boldsymbol{G}_{\boldsymbol{\varepsilon}}$.

Since the ODE system (2) is large-scale and weakly connected, it can be solved quite efficiently on parallel computers by exploiting of parallelism across system [2, 3]. Equation segmentation (ES) method is the most attractive in this context. According to the ES method the right-hand side of (2) is distributed among different processors and calculations of its parts are carried out simultaneously [3]. At the

same time the secondary topology graph $\boldsymbol{G_\varepsilon}$ is equivalent to the computational graph of the right-hand side function (see Sect. 2.2). It allows to reformulate the problem of optimal parallelization of ODE solvers for (2) by means of equation segmentation in terms of the secondary topology graph partitioning problem.

Architecture aware partitioning of computational graphs is a relatively new area of research [4]. Most of existing partitioning algorithms are suitable for homogeneous parallel architectures, but fail to address the limitations imposed by heterogeneity.

Recently available solutions for complicated heterogeneous parallel architectures [4–14] have some significant drawbacks. For instance, architecture aware partitioning methods in JOSTLE [6, 7], SCOTCH [13] and DRUM [9, 10] do not take into consideration heterogeneity in both processor performance and network. This problem is partially solved in PT-SCOTCH [14]. MINIMAX tries to minimize the variance in processor execution time, but it does not take into account the granularity of the parallelized application [8]. PART/PARAPART [11, 12] and PAGRID [5] are free from the above-mentioned disadvantages, but they use edgecut based model of communications which is a poor approximation of the total communication cost. More accurate model was proposed in [4] and implemented in extention of METIS library for architecture aware partitioning. This model is based on the partitioning volume measure which is defined as the total communication volume required by the partition. However, used in [4] communication metrics do not take into account latencies of links between processors. Furthermore, all mentioned partitioning tools do not rely on the well-known fact that the computational cost on vector processors depends drastically on the structure of the partition. Finally, all available graph partitioners works only with a priori known computational graphs. But in our case, structure of the computational graph depends on the space step ε, which should be also estimated during optimal parallelization process. In order to solve the drawbacks and limitations of existing partitioners in the context of the considered optimal parallelization problem, we propose a new problem-oriented approach to parallelize PDE systems on geometrical graphs in an optimal way.

The contributions of this paper are: (a) enhanced metrics for estimation of execution time of numerical solvers for the problem (1); (b) optimization approach that allows to find both an optimal space step and an optimal partitioning. The reminder of the paper is organized as follows. Section 2 discusses the model of optimal parallelization problem. Section 3 contains an overview of proposed optimization approach. A considered approach is implemented in toolbox πDYNO for optimal parallelization of DNOs simulators in PROMOT/DIANA simulation environment [21]. Some modules of πDYNO are still under development, but several results have already been achieved. The architecture of πDYNO is presented in Sect. 4. Finally, we give conclusions and areas of further work in Sect. 5.

2 Optimization problem

2.1 General model

The de facto criteria for comparing quality of different parallel numerical solvers are an accuracy of obtained solutions, as well as total execution time, speedup and efficiency of application. The execution time T_p of the solver for the ODE system (2) depends on the number of iterations n during ODE system integration and on the set of parameters $\boldsymbol{q}$ that define mapping of computational graph onto target parallel system. In addition, structure of the computational graph for the right-hand side of the ODE system (2) depends on the space step width ε. To find the speedup S_p and efficiency E_p, only the execution time T_p and parameters of the target parallel system need to be known.

The accuracy of the solver E can be roughly estimated from the discretization steps in space ε and in time τ [2, 16]. The time step τ for discretization of the ODE system (2) allows to uniquely identify the number of iterations n and conversely. On the other hand discretization steps ε and τ can not vary independently. They are constrained by an inequality that imposes restrictions on the stability domain of the chosen ODE integration method. We refer to this inequality as stability inequality.

As a result, the following model of optimal parallelization problem is proposed

$$\begin{cases} \min_{\boldsymbol{x}\in X} \left\{ \left(T_p(\boldsymbol{x}),\, S_p^{-1}(\boldsymbol{x}),\, E_p^{-1}(\boldsymbol{x}),\, E(\varepsilon,\tau)\right)\cdot \boldsymbol{w} \right\} \\ \text{subject to} \quad T_p(\boldsymbol{x}) \le T_p^{\max} \\ \qquad\qquad f_{\text{sv}}(\varepsilon,\tau) \le 0 \\ \qquad\qquad E(\varepsilon,\tau) \le E^{\max} \end{cases}, \tag{3}$$

where $\boldsymbol{x} = (\varepsilon, \tau, \boldsymbol{q})$ is a tuple of unknowns, $\boldsymbol{w} \in [0,1]^4$ is a vector of weights for optimality criteria, $T_p^{\max}$ and $E^{\max}$ are upper bounds for the execution time and the accuracy of solutions, f_{cv} is a stability violation function which defines stability inequality. Inequality constraints in this model represent the most important restrictions on parallelization. Furthermore, objective function in (3) allows by means of weights vector $\boldsymbol{w}$ to choose and combine the most meaningful metrics for users. The only restriction is that objective function must involve at least one execution time related criterion.

2.2 Execution time related metrics in the general model

The execution time of the ODE integrator depends significantly on the time of right-hand side function evaluation.

The computational graph for the right-hand side of the system (2) can be modeled by the secondary topology graph $\boldsymbol{G_\varepsilon}$. Since sets of equations that correspond to different vertices of $\boldsymbol{G_\varepsilon}$ are computationally equivalent, the computational graph is unweighted.

For modeling the target parallel system we use a weighted undirected graph $\boldsymbol{A} = (V_A, E_A)$. The vertices $p_i \in V_A$ represent processors in the system, and the edges $e_i \in E_A$ correspond to the communication links between processors. Array based computations, as well as communications are modeled by linear functions. Linear models are well suited for representation of operations on vector processor and point-to-point communications. Linear model of communications is widely used and well known in literature as Hockney's model. Thus each vertex and edge of $\boldsymbol{A}$ has associated with it latency and cost per unit operation. The unit computational operation is an evaluation of the right-hand sides of the equations that corresponds to some vertex $v \in V_{G_\varepsilon}$. The unit communication operation is an data transfer from one vertex of $\boldsymbol{G_\varepsilon}$ to another. Cost per unit communication operation between linked processors p_i and p_j is equal to $m/b_w(p_i, p_j)$, where m denotes transferred data size, and $b_w(p_i, p_j)$ denotes bandwidth of the link. For modeling links between processors without explicit edges between them, we use linear path length (LPL) model [6]. It allows to obtain communicational costs for directly unlinked processors.

Given the proposed models of the computational graph and the target parallel system, we now define execution time related metrics. The computational cost for a processor $p_i \in V_A$, i.e. the cost processor p_i incur for processing over all portion of vertices $V_i \subseteq V_{G_\varepsilon}$ assigned to it, is a sum of all array based computations on p_i:

$$\begin{aligned}\mathrm{CompCost}_{p_i}^{V_i} = l_p(p_i)|\{e \in E_{\boldsymbol{\Gamma}}\,; \\ \mathrm{enode}(e) \cap V_i \neq \emptyset\}| + c_p(p_i)|v \in V_i|\,,\end{aligned} \tag{4}$$

where $l_p(p_i)$ denotes the latency of array based computation on p_i, $c_p(p_i)$ is the cost per unit computational operation on p_i, and enode(e) is a set of secondary topology graph nodes, that were inserted into edge e of graph $\boldsymbol{\Gamma}$ on the stage of half-discretization (see Sect. 1). Since communications of a processor $p_i \in V_A$ with another can be performed simultaneously, the cost processor p_i incur for communicating any information associated to V_i is defined as follows:

$$\begin{aligned}\mathrm{CommCost}_{p_i}^{V_i} = \max_{p_j \in V_A \setminus p_i} \big\{ l_e(p_i, p_j) \\ \times |\mathrm{adj}(V_i, V_j) \neq \emptyset| + c_e(p_i, p_j)|\mathrm{adj}(V_i, V_j)| \big\}\,,\end{aligned} \tag{5}$$

where $l_e(p_i, p_j)$ and $c_e(p_i, p_j)$ denote the latency and the cost per unit computation between p_i and p_j, and $\mathrm{adj}(V_i, V_j) = \{v \in V_i; \exists v' \in V_j \wedge e(v, v') \in E_{G_\varepsilon}\}$ denotes a subset of V_i vertices adjacent to vertices in V_j. Figure 1 gives insight into this formula.

Using the above definitions, the execution time of the parallel solver is:

$$\begin{aligned}T_p = n(\tau, \varepsilon) \\ \times \left(t_0(\tau, \varepsilon) + \max_{p_i \in V_A} \left\{ \mathrm{CompCost}_{p_i}^{V_i} + \mathrm{CommCost}_{p_i}^{V_i} \right\} \right),\end{aligned} \tag{6}$$

where t_0 is an average time of auxiliary calculations between two sequential right-hand side function evaluations, n is a number of right-hand side function evaluations, need to be performed during ODE system integration. Values of t_0 and n depends only on the chosen ODE integration method and on the step sizes τ and ε. The execution time of the sequential solver is given by

$$\begin{aligned}T_1 = n(\tau, \varepsilon) \Big(t_0(\tau, \varepsilon) + \big(l_p(p_f) - b_p(p_f)\big)\,|E_{\boldsymbol{\Gamma}}| \\ + b_p(p_f) \left(\frac{L}{\varepsilon} + |V_{\boldsymbol{\Gamma}}| \right) \Big),\end{aligned} \tag{7}$$

where vertex p_f corresponds to the fastest processor, and $L = \sum_{e \in E_{\boldsymbol{\Gamma}}} l(e)$ denotes the total weight of edges in $\boldsymbol{\Gamma}$.

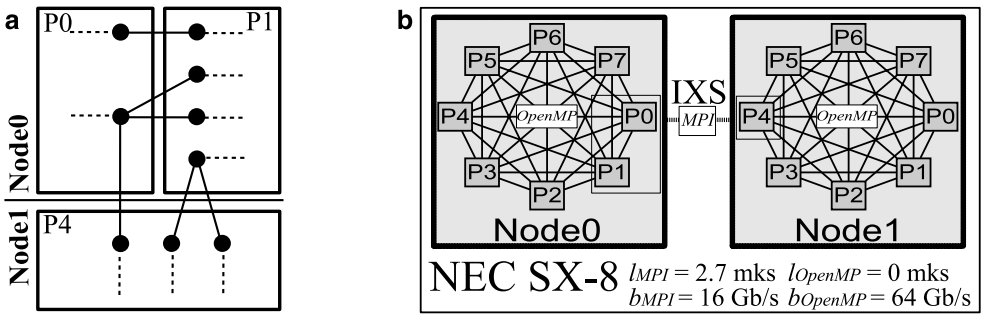

Fig. 1 Estimation of communication cost between processors. The secondary topology graph $\boldsymbol{G_\varepsilon}$ is mapped (**a**) on the processors $P0$, $P1$ (Node0) and $P4$ (Node1) of SX-8 cluster (**b**) [15]. Communication volumes [4] for $P1$ are $Vol(P1, P0) = |\mathrm{adj}(V_{P1}, V_{P0})| = 3$ and $Vol(P1, P4) = |\mathrm{adj}(V_{P1}, V_{P4})| = 1$. Formula $\mathrm{CommCost}_{P1} = \max\left\{l_{\mathrm{open}MP} + Vol(P1, P0)(m/b_{\mathrm{open}MP}), l_{MPI} + Vol(P1, P4)(m/b_{MPI})\right\} = l_{MPI} + m/b_{MPI}$ gives the communication cost for $P1$

Given metrics allows to define the relative speedup S_p and efficiency E_p by the following formulas:

$$S_p = \frac{T_1}{T_p} \quad E_p = \frac{T_1}{PT_p}, \tag{8}$$

where P denotes the peak performance of utilized processors.

2.3 Stability violation and global error estimates

To derive the stability violation function f_{sv} we use the following procedure. At first, we estimate the spectrum of the ODE system Jacobian $S(J_{F_{G_\varepsilon}})$. After that estimates of the spectrum diversity are used to exclude problem dependent terms in the stability function of the ODE integration method $R(\lambda)$ [16]. Hence, expression

$$\left\{(\tau, \varepsilon) \in \mathbb{R}^2; \left| R\left(S^*\left(J_{F_{G_\varepsilon}}\right)\right) = 1\right|\right\}, \tag{9}$$

where S^* is the upper (lower) estimate of the spectrum diversity for explicit (implicit) integration method, gives a set of points with $f_{\text{cv}}(\tau, \varepsilon) = 0$.

For some special classes of ODE systems spectrum $S(J_{F_{G_\varepsilon}})$ can be calculated exactly. Nevertheless, in the majority of cases it is rather complicated, and therefore one need to use some inequalities for eigenvalues. Such inequalities can be derived from standard matrix inequalities, namely Hirsch's, Bendixson's and Brauer's inequalities, corollaries of Ostrowski's theorem [17], or on the basis of particular inferences. In general, these inequalities are not too sharp, but for each inequality there are classes of ODE systems for which this inequality yield good estimate. Therefore, all known estimates should be applied to the given problem and then the best one should be chosen. Some useful inequalities for $S(J_{F_{G_\varepsilon}})$ can be derived on the basis of information about $\boldsymbol{G_\varepsilon}$. Applying eigenvalues for laplacian matrix of the secondary topology graph is the most promising direction in this field [18, 19].

Since reliable a priory estimation of global error usually gives significantly overrated values for nonlinear ODE systems, we use the following formula for expected accuracy of the solver

$$E(\tau, \varepsilon) = a(\boldsymbol{F_{G_\varepsilon}}) E_m\left(\tau, \Phi\left(\boldsymbol{F_{G_\varepsilon}}\right)\right), \tag{10}$$

where E_m is a function of reliable a priory global error estimate for the target ODE integration method, $a(\cdot)$ is a relaxation coefficient, and $\Phi(\cdot)$ is a mapping, that allows to obtain Lipschitz constant for the given function.

3 Optimization approach

The key idea of our optimization approach is in separation of architecture aware partitioning of the secondary topology graph from estimation of the step sizes τ and ε. It is possible due to independence of accuracy and stability constraints in (3) from information about partitioning. We use the nested optimization scheme (see Fig. 2). The "outer" optimizer tries to find optimal values of the step sizes τ and ε. Each iteration of the "outer" optimizer entails three steps. In the first step, a sample points $\boldsymbol{x} = (\tau, \varepsilon)$ is produced. The second step generates partitioning $\boldsymbol{q}$ of the secondary topology graph $\boldsymbol{G_\varepsilon}$ for the given $\boldsymbol{x}$ according to the optimality criterion (6). The third step addresses the calculation of objective function and the treatment of infeasible samples.

Input: E^{max}, T_p^{max}, $\mathbf{w}$, $\boldsymbol{\Gamma}$, $\mathbf{A}$, $X \subset \mathbb{R}^2$, $f_{cv}(\cdot)$, $E(\cdot)$
Output: τ, ε, $\mathbf{q}$
begin
  Initialize statistical global optimizer
  while *Convergence of statistical global optimizer* **do**
    Generate a new sample $\mathbf{x} \in X$ with statistical global optimizer
    if $E(\tau, \varepsilon) \le E^{max}$ *and* $f_{cv}(\tau, \varepsilon) \le 0$ **then** `/* sample x is feasible */`
      Generate a secondary topology graph $\mathbf{G}_\varepsilon = \mathbf{G}(\varepsilon, \boldsymbol{\Gamma})$
      Compute an initial partitioning $\mathbf{q}$ of $\mathbf{G}_\varepsilon$ with available graph partitioner
      Correct the initial partitioning $\mathbf{q}$ with some heuristic `/* see Formula (6) */`
      Tune randomized parameters of heuristic with bayesian heuristic approach
      if $T_p(\tau, \varepsilon, \mathbf{q}) \le T_p^{max}$ **then**
        Calculate objective function `/* see Formula (3) */`
        continue
    Treat sample $\mathbf{x}$ as infeasible
end

Fig. 2 Outline of the optimization approach

In the partitioning step, we use a predictor-corrector approach similar to one proposed by Moulitsas and Karypis in [4]. The partitioning for prediction phase is computed using available graph partitioning library. Architecture aware partitioners are preferred. In this phase, model of the execution time objective (see (6)) is simplified. Namely, computational and communication latencies are not taken into account. The prediction phase allows to leverage existing high-quality partitioning software. The partitioning for correction phase is computed by utilizing some randomized combinatorial optimization heuristic (randomized greedy algorithms, genetic algorithms, ant colony method, simulated annealing, tabu search etc). The objective for this phase is given by (6). After each correction phase we use the Bayesian heuristic approach for tuning randomized parameters of the combinatorial optimizer [20]. It allows to improve convergence rate of the randomized heuristic.

Note that partitioning step is computationally very expensive. Furthermore, the estimated partitioning $\boldsymbol{q}$ depends on some randomized parameters of the combinatorial optimizer. It leads to partial uncertainty of time related metrics and hence to stochastic objective in the general model (3). Therefore "outer" optimizer should realize stochastic black-box optimization approach for expensive objectives. Statistical optimization methods fall under this category [22].

Table 1 Statistical optimization methods in DIANA

Routine	Package	Description
bayes1	GMFL [20]	Bayesian global optimization method
unt	GMFL	The global method of extrapolation type by A. Zilinskas
lbayes	GMFL	The local Bayesian method by J. Mockus
direct	DIRECT v2.0 [23]	Dividing rectangular global optimization method
bbowda	BBOWDA [24]	Black box global optimization method with data analysis

4 Description of toolbox πDyNO for optimal parallelization of DNOs simulators in ProMoT/Diana simulation environment

The proposed parallelization approach is implemented in πDyNO toolbox (the name stands for "Parallelization tool for Dynamic Network Objects Simulators"). πDyNO is realized as supplementary software for ProMoT/Diana simulation environment. Figure 3 shows a flow diagram of πDyNO in ProMoT/Diana. At present πDyNO consists of two shared libraries (`pdynoopt.so` and `pdynopart.so`) and two utilities (`pdyno2bs` and `pdyno2c++`). The library `pdynopart.so` represents optimization problem for the partitioning step. The library `pdynoopt.so` corresponds to general representation of optimization problem for the "outer" optimizer (see Sect. 3). The above-mentioned libraries allows to generate XML file with representation of optimally parallelized solver for the PDE system on geometrical graph. This file is used as an input by `pdyno2bs` and `pdyno2c++` utilities. `pdyno2bs` generates script for running the parallel application on the nodes defined by optimal partitioning. Formats for Portable Batch System (PBS), as well as for batch environments in MPICH and OpenMPI are supported. `pdyno2c++` generates C++ sources for calculating the right-hand side of the ODE system (2). The sources obtained with `pdyno2c++` realize hybrid OpenMP + MPI code.

The whole process of optimal parallelization with πDyNO can be subdivided into three stages. The first stage concerns further specification of the optimization problem. The modeling tool ProMoT allows to obtain model of accuracy and stability constraints in symbolic representation. The command line utility `mdl2diana` translates this model to C++ source files, which can be compiled and linked to a shared library by the utility `dianac`. The compiled model is used as an input by `pdynoopt.so`. The geometrical $\boldsymbol{\Gamma}$ and architecture $\boldsymbol{A}$ graphs should be specified in XML-files, which need to be defined for `pdynoopt.so`.

The second stage deals with running optimal parallelization procedure in the simulation environment DIANA. DIANA is based on the dynamic object-oriented programming language Python and inherits the Python command line user interface. Hence, the user should assign optimization problems from `pdynoopt.so` and `pdynopart.so` with optimizers in Python script, and run this script with `diana` utility. Table 1 lists statistical optimization methods, which can be used as "outer" optimizers in DIANA.

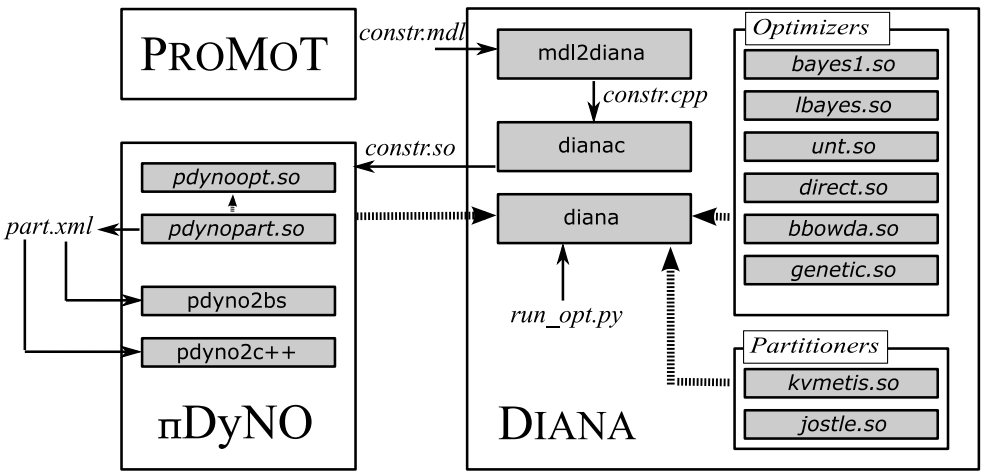

Fig. 3 πDyNO flowchart

In the third stage, utilities `pdyno2bs` and `pdyno2c++` should be run to obtain sources of optimally parallelized solver.

5 Conclusions and further work

In this paper, we address issues in the architecture aware parallelization of solvers for PDE systems on geometrical graphs. We propose and discuss an approach to formalize and solve the optimal parallelization problem. Our approach allows to optimize both the characteristics of discretization method, and the mapping of discretized problem on the target parallel system. Finally, we present a tool, πDyNO, for the semi-automatic parallelization of solvers, which implements this approach.

Further work will consist of developing testing environment for πDyNO, and empirical investigation of parallel PDE solvers obtained with πDyNO on real parallel systems. Since parallelism across system does not restrict the use of parallelism across method and parallelism across time [3], the other direction for further work will be concerned with the development of an approach to optimally parallelize solvers that entail both parallelism across system and parallelism across method.

Acknowledgement Authors would like to thank Process Synthesis and Process Dynamics Research Group of Max Planck Institute for Dynamics of Complex Technical Systems, as well as Process Engineering and Technology Network of Competence – Pro3 for financial and technical support of this work.

References

1. Pokornyj YV, Penkin OM et al. (2005) Differential Equations on Geometrical Graphs. Fizmatlit, Moscow, p 210 (in russian)
2. Hairer E, Nørsett SP, Wanner G (2008) Solving Ordinary Differential Equations I: Nonstiff Problems. Springer, Heidelberg, p 528
3. Petcu D (1998) Parallelism in Solving Ordinary Differential Equations. Tipografia Universitatii, Bucharest, p 232
4. Moulitsas I, Karypis G (2008) Architecture Aware Partitioning Algorithms, In: Proc 8th Int Conf on Algorithms and Architectures for Parallel Processing (ICA3PP), LCNS, vol 5022, pp 42–53
5. Huang S, Aubanel E, Bhavsar V (2006) PaGrid: A mesh partitioner for computational grids. J Grid Comput 4:71–88
6. Walshaw C, Cross M (2001) Multilevel mesh partitioning for heterogeneous communication networks. Future Generat Comput Syst 17:601–623
7. Walshaw C, Cross M (2007) Jostle: Parallel Multilevel Graph-Partitioning Software – An Overview. In: Magoules F (ed) Mesh Partitioning Techniques and Domain Decomposition Techniques, Civil-Comp Ltd, pp 27–58
8. Kumar S, Das SK, Biswas R (2002) Graph Partitioning for Parallel Applications in Heterogeneous Grid Environments, In: Parallel and Distributed Processing Symposium, Proceedings International, IPDPS 2002, pp 66–72
9. Faik J (2005) A Model for Resource-Aware Load Balancing on Heterogeneous and Non-Dedicated Clusters, PhD Thesis, Rensselaer Polytechnic Institute Troy, New York, p 90
10. Teresco JD, Faik J, Flaherty JE (2006) Hierarchical Partitioning and Dynamic Load Balancing for Scientific Computation, In: Applied Parallel Computing, 7th Int Conf PARA 2004 Revised Selected Papers, LNCS, pp 911–920
11. Chen J, Taylor VE (1999) ParaPART: Parallel Mesh Partitioning Tool for Distributed Systems, In: Proc 11th IPPS/SPDP–99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, LNCS, vol 1586, pp 996–1005
12. Chen J, Taylor VE (2002) Mesh partitioning for efficient use of distributed systems, IEEE Trans Parallel Distributed Syst 13(1): 67–79
13. Pellegrini F, Roman J (1996) Scotch: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs, In: Proc Int Conf and Exhibition HPCN Europe 1996, LNCS, pp 493–498
14. Chevalier C, Pellegrini F (2008) PT-Scotch: A tool for efficient parallel graph ordering, Parallel Comput 34(6–8):318–331
15. Tagaya S et al. (2006) The NEC SX-8 Vector Supercomputer System. In: Resch M, Bonisch T, Benkert K (eds) High Performance Computing on Vector Systems. Springer, Berlin, Heidelberg, pp 3–24
16. Hairer E, Wanner G (2004) Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems. Springer, Berlin, p 614
17. Markus M, Mink H (1992) A Survey of Matrix Theory and Matrix Inequalities. Courier Dover Publications, New York
18. Doob M (2003) Spectral Graph Theory. In: Gross JL, Yellen J (eds) Handbook of Graph Theory, CRC Press, Boca Raton, pp 557–574
19. Cvetković D, Doob M, Sachs H (1995) Spectra of Graphs: Theory and Application. Ambrosius Barth Verlag, Heidelberg, Leipzig, p 448
20. Mockus J (2000) A Set of Examples of Global and Discrete Optimization: Applications of Bayesian Heuristic Approach. Springer, Dordrecht, p 321
21. Krasnyk M (2008) Diana – An object-oriented Tool for Nonlinear Analysis of Chemical Processes/Forschungsberichte aus dem Max-Planck-Institut für Dynamik komplexer technischer Systeme 23, Shaker, Aachen, p 129
22. Zhigljavsky A, Zilinskas A (2007) Stochastic Global Optimization. Springer, New York, p 272
23. Finkel DE (2003) DiRect Optimization Algorithm User Guide: Technical Report, NC 27695-8205, North Carolina State University, Raleigh, North Carolina, p 14
24. Kofler K (2007) Black Box Optimization with Data Analysis: Diploma thesis, Universität Wien, Wien, p 143, http://www.tigen.org/kevin.kofler/bbowda/. Accessed 27 April 2009