Journal of Computational Science 4 (2013) 377–385

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Efficient SIMD solution of multiple systems of stiff IVPs

Andrew Kroshko, Raymond J. Spiteri*

Department of Computer Science, University of Saskatchewan, Saskatoon, Saskatchewan S7N 5C9, Canada

ARTICLE INFO

Article history:
Received 22 March 2012
Received in revised form 24 August 2012
Accepted 27 August 2012
Available online 7 September 2012

Keywords:
Ordinary differential equations
Stiff equations
Parallel computing
Chemical reactions
CO_2 reforming
Plug flow reactor
Cell broadband engine

ABSTRACT

The parallel solution of multiple systems of initial-value problems (IVPs) in ordinary differential equations is challenging because the amount of computation involved in solving a given IVP is generally not well correlated with that of solving another. In this paper, we describe how to efficiently solve multiple systems of stiff IVPs in parallel within a single-instruction, multiple-data (SIMD) implementation on the Cell Broadband Engine (CBE) of the RODAS solver for stiff IVPs. We solve two systems of stiff IVPs simultaneously on each of the eight synergistic processing elements per CBE chip for a total of 16 systems of IVPs. We demonstrate a speedup of 1.89 (a parallel efficiency of over 94%) over the corresponding serial code on a realistic example involving the operation of a chemical reactor. The techniques described apply to other multi-core processors besides the CBE and can be expected to increase in importance as computer architectures evolve to feature larger word sizes.

1. Introduction

Many important physical processes are described by initial-value problems (IVPs) in ordinary differential equations (ODEs). These IVPs must typically be solved numerically, and for the purposes of obtaining solutions as quickly as possible, a great deal of attention is being given to algorithms that take advantage of parallel computing. We are interested in the simultaneous solution of systems of IVPs, e.g., systems of ODEs that are to be solved with different parameter values, initial conditions, etc. The approach we take is based on the paradigm known as *single instruction multiple data* (SIMD) [1], a form of parallelism in which multiple pieces of data are processed simultaneously by the same instruction. Although it might appear at first that solving systems of IVPs simultaneously is easily amenable to massive multiple-instruction, multiple-data parallelization, the amount of computation associated with solving a given IVP is generally not well correlated with that of solving another. Consequently a straightforward implementation of such an algorithm would likely suffer from poor load balancing and not perform well. Furthermore at present such architectures are not commonly available. On the other hand, SIMD is a common feature on modern processors, such as graphics processing units (GPUs), the Intel x86 architecture, where it is called streaming SIMD extensions, and the Cell Broadband Engine (CBE), where in fact it is the only mode of operation on the synergistic processing elements (SPEs). The CBE is currently used in scientific computing on both large [2–4] and small scales [5,6] due to its high floating-point throughput. The CBE allows SIMD instructions to be used without resorting to assembly language and provides a great deal of programmer control over memory management. The SPEs do not have features such as instruction reordering that are typical of other architectures; therefore the number of clock cycles to execute any program is highly predictable, making the CBE ideal for controlled experiments with algorithms involving SIMD [7]. This also makes the CBE ideal for the study of algorithms that take advantage of new computer architectures. SIMD is often applied in situations in which there are several independent problems that share a common solution algorithm applied to different data. However, we show that SIMD can also be applied in less straightforward situations, e.g., when the solution algorithm has parts that execute different instructions depending on the data.

We implement a RODAS solver [8] to exploit SIMD instructions when solving multiple systems of stiff IVPs simultaneously. The numerical solution of individual IVPs is also in general not easily parallelizable, especially within the SIMD paradigm. One immediate issue faced is that the number of steps taken by variable step-size IVP solvers is problem dependent, thus reducing the effectiveness of SIMD instructions. A second issue is that the standard algorithm used for step-size control is also not conducive to the use of SIMD instructions due to branching. We do not address these issues here. Rather we use a mathematical model for the kinetics involved in CO_2 reforming [9,10] to demonstrate that these issues affect only a small part of the overall computational effort within a SIMD strategy for the parallel solution of multiple systems of IVPs.

* Corresponding author.
E-mail address: spiteri@cs.usask.ca (R.J. Spiteri).

1877-7503/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jocs.2012.08.017

The remainder of the paper is organized as follows. In Section 2, we give a description of the IVP solver RODAS and our SIMD implementation of it to simultaneously solve multiple IVPs. In Section 3, we briefly describe an example on which we test the SIMD RODAS solver. The example involves the operation of a chemical reactor. In Section 4, we demonstrate the effectiveness of our implementation on this example. Finally, in Section 5 we offer some conclusions from this study.

2. A SIMD Rosenbrock solver for multiple stiff IVP systems

To illustrate the SIMD solution of multiple systems of stiff IVPs, we choose the Rosenbrock method from the RODAS solver [8]. This method is fourth-order accurate with a third-order embedded method for error and step-size control. Error and step-size control are generally considered essential for the efficiency and reliability of IVP solvers. These solvers use a local error estimate to take the largest step sizes possible while still meeting a user-specified error tolerance. In practice, the steps used by constant step-size solvers are constrained by the most difficult regions of the problem, ultimately leading to unacceptably large errors or long computation times. A further advantage of Rosenbrock methods is that, unlike general fully implicit Runge–Kutta methods for which nonlinear equations must be solved for each stage, only a system of linear equations must be solved for each stage. Thus a linearly implicit Runge–Kutta method is typically only moderately more complex to implement than an explict Runge–Kutta method [8]. This form of implicitness is of further advantage for SIMD implementations because the direct solution of linear systems of a given size requires a highly predictable number of operations. However, Rosenbrock methods are not generally as stable as fully implicit methods. In practice, this means that the integration may require smaller steps than it would with a fully implicit method. Nonetheless, the overall trade-off in terms of computation time is often still favourable because each step of a Rosenbrock method is much less expensive. The specific Rosenbrock method in RODAS was designed to satisfy additional order conditions for differential-algebraic equations in order to enhance its stability [8].

To simplify the presentation but without loss of generality, we consider IVPs in *autonomous* form

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \quad t_0 < t < t_f, \quad \mathbf{y}(t_0) = \mathbf{y}_0, \tag{1}$$

where $\mathbf{y}$ is an m-vector describing the state of the system at time t and the right-hand side function $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^m$ does not depend explicitly on the independent variable t. This also allows several simplifying assumptions in the implementation of RODAS solvers.

An s-stage Rosenbrock method to advance a numerical solution of (1) from $\mathbf{y}_n \approx \mathbf{y}(t_n)$ at $t = t_n$ to $\mathbf{y}_{n+1} \approx \mathbf{y}(t_{n+1})$ at $t_{n+1} = t_n + \Delta t_n$ is given by

$$\mathbf{k}^{(i)} = \Delta t_n \mathbf{f}\left(\mathbf{y}_n + \sum_{j=1}^{i-1} \alpha_{ij}\mathbf{k}^{(j)}\right) + \Delta t_n \mathbf{J}_n \sum_{j=1}^{i} \gamma_{ij}\mathbf{k}^{(j)}, \quad i = 1, 2, \ldots, s, \tag{2a}$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \sum_{i=1}^{s} \beta_i \mathbf{k}^{(i)}, \tag{2b}$$

where $\mathbf{k}^{(i)}$ is stage i of the current step, α_{ij} are the coefficients for the explicit part of the method, γ_{ij} are the coefficients for the (linearly) implicit part of the method, and β_i are the so-called quadrature weights. Here, $\mathbf{J}_n$ is the Jacobian matrix of partial derivatives of $\mathbf{f}$ with elements defined by

$$\mathbf{J}_n(i,j) = \frac{\partial f_i}{\partial y_j}, \quad i, j = 1, 2, \ldots, m, \tag{3}$$

where f_i is component i of $\mathbf{f}$ and y_j is component j of $\mathbf{y}$ in (1), evaluated at $\mathbf{y}_n$. Each $\mathbf{k}^{(i)}$ is determined from the solution of a system of linear equations with the coefficient matrix $(\mathbf{I} - \Delta t\, \gamma_{ii} \mathbf{J}_n)$. The Rosenbrock method used in RODAS chooses $\gamma_{11} = \gamma_{22} = \ldots = \gamma_{ss} = \gamma$, and thus only requires one LU-decomposition per step instead of one per stage.

Direct implementation of (2) also involves matrix-vector multiplications $\mathbf{J}_n \mathbf{k}^{(j)}$, $j = 1, 2, \ldots, i-1$, at stage i. We thus introduce new variables

$$\mathbf{u}^{(i)} = \sum_{j=1}^{i} \gamma_{ij}\mathbf{k}^{(j)}, \quad i = 1, 2, \ldots, s.$$

If $\gamma_{ii} \neq 0$, $i = 1, 2, \ldots, s$, then the lower-triangular matrix $\mathbf{\Gamma}$ with elements $\mathbf{\Gamma}(i,j) = \gamma_{ij}$ is invertible, and we can write

$$\mathbf{k}^{(i)} = \frac{1}{\gamma_{ii}}\mathbf{u}^{(i)} - \sum_{j=1}^{i-1} g_{ij}\mathbf{u}^{(j)}, \tag{4}$$

where g_{ij} is the (i,j) element of the matrix

$$\mathbf{G} = \mathrm{diag}(\gamma_{11}^{-1}, \ldots, \gamma_{ss}^{-1}) - \mathbf{\Gamma}^{-1}.$$

Using (4) in (2) yields

$$\left(\frac{1}{\Delta t_n\, \gamma_{ii}}\mathbf{I} - \mathbf{J}_n\right)\mathbf{u}^{(i)} = \mathbf{f}\left(\mathbf{y}_n + \sum_{j=1}^{i-1} a_{ij}\mathbf{u}^{(j)}\right) + \sum_{j=1}^{i-1}\left(\frac{g_{ij}}{\Delta t_n}\right)\mathbf{u}^{(j)}, \quad i = 1, 2, \ldots, s, \tag{5a}$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \sum_{j=1}^{s} b_j \mathbf{u}^{(j)}, \tag{5b}$$

where a_{ij} are the coefficients of the explicit part of the new formula that are given by element (i,j) of the matrix $\mathcal{A}\,\mathbf{\Gamma}^{-1}$ with $\mathcal{A}(i,j) = \alpha_{ij}$, $g_{i,j}$ are the coefficients for the linearly implicit part of new formula, and b_j are the quadrature weights of the new formula that are given by component j of the vector $\mathcal{B}\,\mathbf{\Gamma}^{-1}$ with $\mathcal{B}(i) = \beta_i$. This form of a Rosenbrock method avoids matrix-vector multiplications and hence is more efficient to implement.

In the RODAS solver, the main method, which is used to advance the numerical solution, is fourth order and contains $s = 6$ stages. The RODAS solver also has a 5-stage embedded method that produces a result of third order. For efficiency, the embedded method also provides the explicit part of stage 6 of the main method; i.e., no further evaluations of $\mathbf{f}(\mathbf{y})$ are required before solving the linear system to obtain $\mathbf{k}^{(6)}$. The main and embedded methods have different values for the quadrature weights b_j in (5b).

Error and step-size control are implemented as follows. The local error estimate at time t_{n+1} is defined as

$$\mathbf{e}_{n+1} = \mathbf{y}_{n+1} - \hat{\mathbf{y}}_{n+1},$$

where $\mathbf{y}_{n+1}, \hat{\mathbf{y}}_{n+1}$ are the results from the main and embedded methods, respectively. In the RODAS solver, $\mathbf{e}_{n+1}$ is generated directly and added to $\hat{\mathbf{y}}_{n+1}$ to calculate $\mathbf{y}_{n+1}$. A mixed error tolerance at time t_{n+1} is computed for solution components, $i = 1, 2, \ldots, m$, according to

$$\tau_{n+1,i} = \tau_{\mathrm{abs},i} + \tau_{\mathrm{rel},i}\,|y_{n+1,i}|,$$