Unified GPU-Parallelizable Robot Forward Dynamics Computation Using Band Sparsity

Yajue Yang, Yuanqing Wu, and Jia Pan

***Abstract*—This letter proposes a unified GPU-parallelizable approach for robot forward dynamics (FD) computation based on the key fact that parallelism of prevailing FD algorithms benefits from the essential band sparsity of the joint space inertia (JSI) matrix or its inverse. The existing FD algorithms are categorized into three classes: direct JSI algorithms, propagation algorithms, and constraint force algorithms. Their associated systems of linear equations are transformed into a set of block bidiagonal (the first and second classes) and tridiagonal (the third class) linear systems, which can be conveniently and efficiently parallelized over the existing CPU–GPU programming platforms using various state-of-the-art parallel algorithms, such as parallel all-prefix sum (scan) and odd–even elimination. This high-level perspective allows unified and efficient implementation of all three classes of algorithms and also other potentially efficient algorithms, with the bonus that different algorithms can be swiftly compared to recommend problem-specific solutions.**

***Index Terms*—Dynamics, GPGPU, parallel computing, redundant robots.**

I. Introduction

In ROBOT simulation, the acceleration of the robot system must be computed for any applied force (forward dynamics, or FD) [1]. Albeit being a thoroughly explored subject, fast FD computation remains a relevant issue in emerging subfields of robotics. For example, while hyper-redundant robots for dexterous manipulation provide opportunities for working in complex environment [2], their large degree of freedoms (DOF) poses new challenges to fast simulation. In addition, given a kinematic reference trajectory of a robot, we can generate dynamically feasible trajectories using the concept of dynamics filter proposed in [3], by largely evaluating dynamics with sampled states around the reference and different control inputs. An optimal trajectory can then be selected [4]. Such applications involving a vast number of FD calculations also require high computational efficiency and motivate our current research on accelerating FD computation using CPU-GPU platform.

Prevailing FD algorithms can arguably be categorized into three classes. The first class involves directly calculating the joint space inertia (JSI) matrix and then its inversion or factorization using numerical methods, such as the *JSI inversion algorithm* (JSIIA) proposed in [5]. However, this approach has a computational complexity of $O(n^3)$ in the worst case where n denotes the DOF, and therefore unsuitable for robots with large n. The second class performs propagation of velocity, acceleration and force along kinematic chains of a serial or tree-structured robot, leading to recursive sequential $O(n)$ algorithms [1], a typical example being the *articulated-body inertia algorithm* (ABIA) proposed by Featherstone [6]. By approaching from a linear estimation perspective, Rodriguez *et al.* eventually unified the propagation algorithms and the JSI inversion methods by considering the recursions of the former as iterative factorizations of the JSI in the latter [7]. Meanwhile, the third class explicitly computes interbody constraint forces of robots in comparison to direct elimination of constraint forces in the first two classes [8], [9].

Many attempts have been made to fully or partially parallelize the aforementioned three classes of FD algorithms [10]–[13]. Since the constraint force matrix of a serial robot is block tri-diagonal [8], Fijany *et al.* proposed to parallelize it via the *odd-even-elimination-algorithm* (OEEA) and named it the *constraint force algorithm* (CFA) [14]. It leads to a different factorization of the inverse JSI from that in the propagation algorithms. Recently, YANG *et al.* utilized the *all-prefix-sums (scan)* operation in parallelizing the linear recursions involved in both the JSIIA and the ABIA [15]. In comparison to the CFA, the scan-based parallel/hybrid JSIIA and ABIA could only achieve sup-$O(\log n)$ time complexity due to the low degree of parallelism in the inverse of the JSI matrix and the nonlinear recursive computation of the ABIs. In the former case, parallelized inversion of a general matrix may achieve a complexity of $O(\log^2 n)$ [16]. In the latter case, the nonlinear recursion cannot speed up beyond a constant factor [14]. Nevertheless, this does not necessarily imply that scan-based JSIIA and ABIA are always outperformed by truly $O(\log n)$ parallel algorithms (such as the CFA) in all scenarios, because the computation cost of both ABIA and JSIIA (for robot with small number of links) are lower than that of the CFA [14]. These observations justify yet another investigation to identify the application par excellence for scan-based parallel and hybrid algorithms.

Manuscript received February 15, 2017; accepted July 11, 2017. Date of publication August 3, 2017; date of current version August 25, 2017. This letter was recommended for publication by Associate Editor A. Mueller and Editor K. Lynch upon evaluation of the reviewers' comments. This work was partially supported by NVidia Corp. and HKSAR Research Grants Council (RGC) General Research Fund, CityU 21203216, NSFC/RGC Joint Research Scheme CityU103/16, and CityU Startup Grants 7200485 and 9610349. *(Corresponding author: Jia Pan.)*

Y. Yang and J. Pan are with the Department of Mechanical and Biomedical Engineering, City University of Hong Kong, Hong Kong (e-mail: yajuyang@cityu.edu.hk; jiapan@cityu.edu.hk).

Y. Wu is with the University of Bologna, 40126 Bologna, Italy (e-mail: yuanqing.wu@unibo.it).

Digital Object Identifier 10.1109/LRA.2017.2735479

In this letter, we systematically investigate the parallel JSIIA, the sequential-parallel hybrid ABIA and the parallel CFA using a unified coordinate-invariant Lie group formulation [17]–[19]. We emphasize that the key to parallelization of FD algorithms is to identify and utilize the band-sparsity of JSI and/or its inverse via appropriate factorization. The large system of linear equations of the original FD problem is transformed into a set of block bi-diagonal and tri-diagonal linear systems, which can be substantially accelerated by state-of-the-art parallel algorithms, such as parallel scan and parallel OEEA. By recognizing the band-sparsity of the three categories of algorithms, we may generalize the parallelization strategies from the three representative algorithms to wider variety of algorithms. Furthermore, this high-level perspective serves as a guideline for efficient parallelization of other potential FD algorithms [20]. To test running performances of the three representative algorithms, we implement them on an user-friendly Nvidia CPU-GPU heterogeneous platform [21].

This letter is organized as follows. Section II revisits the scan-based parallel/hybrid JSIIA and ABIA with an emphasis on their associated block bi-diagonal systems, and then reviews the parallel scan operation for their efficient solution. Section III investigates the fully parallel CFA, with an emphasis on its associated block tri-diagonal system. Section IV illustrates a parallel OEEA tailored for solving the block tri-diagonal system associated with the CFA. Section V implements the three representative FD algorithms on the aforesaid platform and tests their performances under different application scenarios. Finally, Section VI discusses the scalability of the presented algorithms for closed-loop systems and also future work.

II. Robot Dynamics Algorithms Involving Block Bidiagonal Systems

Throughout this letter, we shall use the same Lie group notation as in [22] and [17], and also use the global matrix representation [18] for the equations of motion (EOM) of rigid bodies and articulated robots.

A. Recursive Formulation of Robot Dynamics

Recall that the Newton-Euler EOM for a single rigid body is given by [17]:

$$J\dot{V} - \mathrm{ad}_V^*(JV) = F, \tag{1}$$

where the velocity twist V and the applied external wrench F are 6-D vectors; the inertia tensor J denotes a 6×6 symmetric positive-definite matrix; the dual adjoint map ad_V^* may be simply identified with the transpose of 6×6 adjoint matrix ad_V. Equation (1) is also referred as the Euler-Poincaré equation of the special Euclidean group [23].

Consider an open-chain articulated robot with n links. Its EOM can be derived recursively from Eq. (1), which directly leads to a recursive inverse dynamics (ID) algorithm to calculate joint torques for producing prescribed accelerations. We summarize the recursive EOM in the following global matrix form [18]:

$$(I - \Gamma)V = S\dot{q} + \text{const.} \tag{2a}$$

$$(I - \Gamma)\dot{V} = S\ddot{q} + \mathrm{ad}_{S\dot{q}}(\Gamma V) + \text{const.} \tag{2b}$$

$$(I - \Gamma)^T F = J\dot{V} + \mathrm{ad}_V^T(JV) + \text{const.} \tag{2c}$$

$$\tau = S^T F \tag{2d}$$

where $V = (V_1^T, \ldots, V_n^T)^T$ and $\dot{V}$ denote the ensemble vectors of the link velocities and accelerations. τ, q, $\dot{q}$ and $\ddot{q}$ denote, respectively, vectors of the joint torques, positions, velocities and accelerations, for example $\tau = (\tau_1, \ldots, \tau_n)^T \in \mathbb{R}^{n\times 1}$; $S \in \mathbb{R}^{6n\times n}$, $J \in \mathbb{R}^{6n\times 6n}$, $\mathrm{ad}_{S\dot{q}} \in \mathbb{R}^{6n\times 6n}$ and $\mathrm{ad}_V^T \in \mathbb{R}^{6n\times 6n}$ are block diagonal matrices whose diagonal blocks are S_i, J_i, $-\mathrm{ad}_{S_i\dot{q}_i}$ and $-\mathrm{ad}_{V_i}^T$ respectively; $I \in \mathbb{R}^{6n\times 6n}$ denotes the identity matrix and $\Gamma \in \mathbb{R}^{6n\times 6n}$ is given by

$$\Gamma = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ \mathrm{Ad}_{f_{1,2}^{-1}} & 0 & \cdots & 0 & 0 \\ 0 & \mathrm{Ad}_{f_{2,3}^{-1}} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \mathrm{Ad}_{f_{n-1,n}^{-1}} & 0 \end{bmatrix}.$$

The constant terms in Eq. (2) correspond to either the velocity and acceleration of the base link or the force applied at the end-effector.

We emphasize that the block lower bi-diagonal matrix $(I - \Gamma)$ (see [18] for more details) is the forward propagation matrix of link velocities V_i's and accelerations $\dot{V}_i$'s from the base to the end-effector. Dually, the block upper bi-diagonal matrix $(I - \Gamma)^T$ is the backward propagation matrix of external forces F_i's from the end-effector to the base. Fig. 1(a) illustrates the ID algorithm as sequentially solving block bi-diagonal linear systems. The application of the ID algorithm to FD algorithms will be explained in Section II-B.

B. The JSIIA and the ABIA

The EOM of an articulated robot under joint coordinates can be derived by eliminating V, $\dot{V}$ and F from Eq. (2a)–(2d) [22]:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + N(q,\dot{q}) = \tau \tag{3}$$

where $M(q) = S^T(I - \Gamma)^{-T} J (I - \Gamma)^{-1} S$ denotes the JSI matrix. As a result, one can simply solve the unknown joint acceleration $\ddot{q}$ by inverting the JSI:

$$\ddot{q} = M(q)^{-1}(\tau - (\underbrace{C(q,\dot{q})\dot{q} + N(q,\dot{q})}_{\tau^{\text{bias}}})) \tag{4}$$

Note that the bias torque τ^{bias} can be conveniently evaluated using an ID solver by setting $\ddot{q} = 0$ [1]. We shall hereafter denote $\tau - \tau^{\text{bias}}$ by τ^{δ}.

The FD algorithm based on Eq. (4) leads to the JSIIA. In particular, we can compute column i of $M(q)$ by solving an ID problem with $\ddot{q}_i$ set to 1 and $\dot{q}$ and all $\ddot{q}_j$'s $(j \neq i)$ set to zero [5]. Consequently, the calculation of the JSI is transformed into a data-independent group of block bi-diagonal systems.

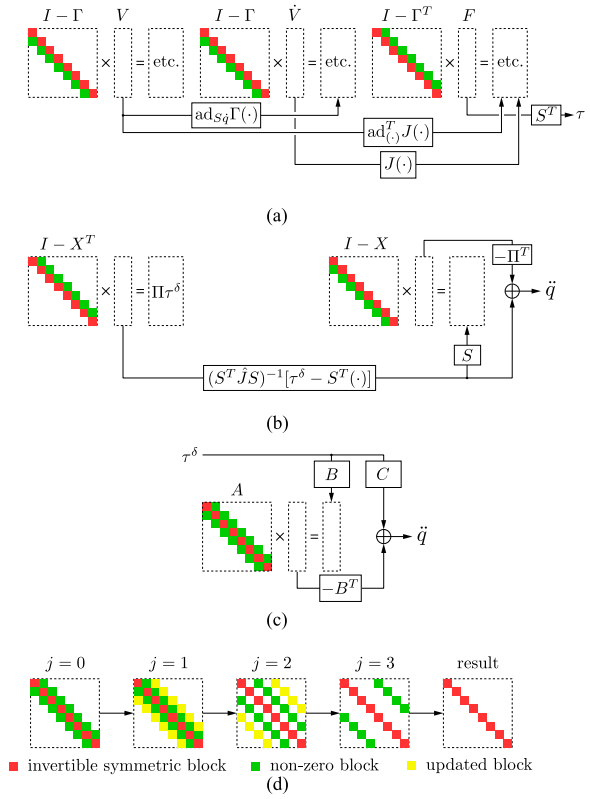

Fig. 1. (a) Inverse dynamics. (b) ABIA. (c) CFA. (d) Diagonalization of an 8×8 block tridiagonal matrix in the OEEA.

The algorithm then proceeds with numerically inverting the JSI matrix or solving the equivalent linear system [1].

Alternatively, the ABIA resorts to computing the ABI of each link that represents the equivalent inertia of a disassembled subsystem. Assuming first that ABIs and bias torques have been computed, the ABIA consists of two linear recursions: (1) backward propagation for computing bias interbody forces produced by succeeding links; (2) forward propagation for computing link accelerations [1], [24]. Finally, joint accelerations are computed using Eq. (2b). This recursive framework of FD computation is due to the following square factorization of $M(q)^{-1}$ [18]:

$$M(q)^{-1} = [I - S^T Y \Pi]^T (S^T \hat{J} S)^{-1} [I - S^T Y \Pi] \quad (5)$$

where $\hat{J}$ is the block diagonal matrix formed by the ABIs of consecutive sub-chains from the last link all the way back to the first link, and can be evaluated by a nonlinear recursion [18]; $\Pi \in \mathbb{R}^{6n \times n}$ has only non-zero block matrices in the upper diagonal where $\Pi_{k,k+1} = \mathrm{Ad}^T_{f^{-1}_{k,k+1}} \hat{J} S_{k+1} (S^T \hat{J} S)^{-1}$. $Y = (I - X^T)^{-1}$ and X is a $6n \times 6n$ block bi-diagonal matrix. This implies that we can calculate $M(q)^{-1}$ by solving the block bi-diagonal systems with $(I - X^T)$ or its transpose being the coefficient matrix. The organization of the block bi-diagonal systems within the ABIA is depicted in Fig. 1(b).

Jain has demonstrated that all recursive $O(n)$ algorithms can be unified into the same two-phase propagation framework with ABI-like inertia matrix [24]. In other words, from an algebraic perspective, computations of τ^{bias}, interbody bias forces and link accelerations in all recursive algorithms basically can be considered as solving block bi-diagonal linear systems.

C. Solving Bidiagonal Systems With Parallel Scan Operations

The aforementioned block bi-diagonal systems can be efficiently solved by several well-known parallel algorithms such as the *recursive doubling algorithm* (RDA) [25] and the scan operation [26]. We shall only consider the latter in this letter.

A *scan operation*, upon specifying an associative binary operator $\oplus$ with identity and an ordered set $[a_0, a_1, \ldots, a_{n-1}]$ of n elements, returns the vector $[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1})]$ [26]. It can be extended to the following linear recursion:

$$x_i = \begin{cases} c_0 & \text{if } i = 0 \\ b_{i-1} x_{i-1} + c_i & \text{if } 0 < i < n \end{cases} \quad (6)$$

with the corresponding bi-diagonal system:

$$\begin{bmatrix} 1 & & & \\ -b_0 & 1 & & \\ & \ddots & \ddots & \\ & & -b_{n-1} & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix} \quad (7)$$

by setting a_i to be the homogeneous matrix

$$a_0 = \begin{bmatrix} 1 & c_0 \\ 0 & 1 \end{bmatrix} \qquad a_i = \begin{bmatrix} b_{i-1} & c_i \\ 0 & 1 \end{bmatrix}, 0 < i < n \quad (8)$$

and $a_{i-1} \oplus a_i$ being the matrix multiplication $a_i a_{i-1}$. Furthermore, a bi-diagonal system such as in Eq. (2a)–(2c) can be easily transformed into scan-compatible forms [15], [26]. An $O(\log n)$-complexity parallel implementation of scan operation (running on n processors) can be achieved via a balanced binary tree structure [26].

Parallelization of the recursive FD algorithms is mainly hindered by the nonlinear recursive computation of the ABIs or their4 equivalents [24]. Fortunately, computation cost can be reduced via hybrid implementation on heterogeneous computing platforms. Taking the CPU-GPU platform for example, the time spent on sequentially computing ABIs on CPU can be partially hidden behind simultaneous calculation of τ^{bias} on GPU.

III. The CFA Involving Block Tridiagonal Systems

The CFA [14] directly calculates the global interbody force F by eliminating the global acceleration $\ddot{q}$, leading to a different factorization of $M(q)^{-1} = C - B^T A^{-1} B$, where A is the constraint force matrix. For open-chain systems, A is a block tri-diagonal matrix and the CFA becomes fully parallelizable.

A. Elimination Strategy With Explicit Constraint Forces

We summarize a general elimination approach compatible with the CFA as follows. First, we reduce Eq. (2a)–(2d) to:

$$(I-\Gamma)\dot{V} = S\ddot{q} \tag{9a}$$

$$(I-\Gamma)^T F = J\dot{V} \tag{9b}$$

$$\tau^\delta = S^T F \tag{9c}$$

by canceling out the bias torque related terms. More specifically, $\tau^\delta = M(q)\ddot{q}$ can be regarded as the required joint torques corresponding to desired joint accelerations of a system with zero velocity. Therefore, the body velocity related terms are not present in Eq. (9a) and (9b). Next, Eq. (9a) and (9b) lead to:

$$S\ddot{q} = (I-\Gamma)J^{-1}(I-\Gamma)^T F \tag{10}$$

Considering a robot with 1-DOF joints, we eliminate $\ddot{q}$ by defining a proper projection matrix $W \in \mathbb{R}^{6n\times 5n}$ such that:

$$W^T S\ddot{q} = W^T(I-\Gamma)J^{-1}(I-\Gamma)^T F = 0 \tag{11}$$

Since Eq. (11) holds for any $\ddot{q}$, $W^T S = 0$. Equation (11) and (9c) should fully determine F, i.e. the following co-efficient matrix $H \in \mathbb{R}^{6n\times 6n}$ must be non-singular:

$$\underbrace{\begin{bmatrix} W^T(I-\Gamma)J^{-1}(I-\Gamma)^T \\ S^T \end{bmatrix}}_{H} F = \tau^\delta \tag{12}$$

Once F is fully determined, $\ddot{q}$ can be derived by pre-multiplying S^T on both sides of Eq. (10):

$$S^T S\ddot{q} = \ddot{q} = S^T(I-\Gamma)J^{-1}(I-\Gamma)^T F \tag{13}$$

where we assume $S^T S = I$ without loss of generality.

B. The Constraint Force Algorithm

Following the notation of [14], we denote $W = \text{diag}(W_1, \ldots, W_n)$ with $W_i \in \mathbb{R}^{6\times 5}$ being a conveniently chosen basis matrix for the constraint force of the ith joint. This results in a decomposition of F into active forces along S and constraint forces F_c along W [14]:

$$F = S\tau^\delta + WF_c \tag{14}$$

which effectively replaces Eq. (9c). It can be shown, with such a choice of W, H is invertible. Substituting Eq. (14) into Eq. (11) leads to:

$$\begin{aligned} AF_c &= -B\tau^\delta \\ A &= W^T(I-\Gamma)J^{-1}(I-\Gamma)^T W \\ B &= W^T(I-\Gamma)J^{-1}(I-\Gamma)^T S \end{aligned} \tag{15}$$

where A is invertible, because the following matrix is non-singular:

$$H\begin{bmatrix} W & S \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \tag{16}$$

Algorithm 1: CALCFWDDYN_CFA($[\tau_i^{\text{in}}], [q_i], [\dot{q}_i]$).

```
/* Compute the bias torques.                          */
[τ_i^bias] ← CALCINVDYN([q_i], [q̇_i], 0)
/* Compute the differential torques in
   parallel.                                          */
[τ^δ] ← CALCDIFFTORQUES([τ_i^in], [τ_i^bias])
/* Parallel computations.                             */
begin
    A ← CALCA([J_i], [Ad_i], [W_i])
    B ← CALCB([J_i], [Ad_i], [W_i], [S_i])
    C ← CALCC([J_i], [Ad_i], [S_i])
    [R_i] ← −Bτ
end
/* Solve a block tridiagonal (BTD)
   system for constraint forces.                      */
[F_{i,c}] ← SOLVBTDSYS(A, [R_i])
/* Compute joint accelerations.                       */
[q̈_i^out] ← CALCACC([τ^δ], [F_{i,c}], B, C)
```

Substitute Eq. (15) back into Eq. (10):

$$\begin{aligned} \ddot{q} &= \underbrace{S^T(I-\Gamma)J^{-1}(I-\Gamma^T)(S}_{C} + W\underbrace{(-A^{-1})B)\tau^\delta}_{F_c} \\ &= \underbrace{(C + B^T(-A^{-1})B)}_{M(q)^{-1}}\tau^\delta \end{aligned} \tag{17}$$

The above equation exhibits a new factorization of $M(q)^{-1}$ Schur complement form [14].

Similar to the CFA, algorithms based on coarse grain parallelism [9], [27] also explicitly solve constraint or interbody forces. As long as the corresponding constraint force or interbody force matrix is band-sparse, these algorithms can achieve full parallelism.

C. Implementation of the Parallel CFA

The parallel CFA first calculates, using the ID algorithm, the bias torques to be deducted from the input torques. The matrix $M(q)^{-1}$ is factored into $M(q)^{-1} = C - B^T A^{-1} B$ as shown in (17). Calculation of A, B, and C is mutually independent and only involves precomputed (by the ID algorithm) constant joint inertia, twists and adjoint transformations. We can compute A, B and C in parallel by exploiting the parallelism of many-core GPU. After solving the block tri-diagonal system in Eq. (15) for constraint forces, the desired joint acceleration vector can be computed through $\ddot{q} = C\tau^\delta + B^T F_c$ which is automatically block-parallelizable. The detailed implementation of the parallel CFA is illustrated in Algorithm 1. Fig. 1(c) illustrates the CFA as a system of linear equations with a block tri-diagonal coefficient matrix.

IV. Parallel Solver for Block Tridiagonal System

In comparison to other matrix multiplications involved in Eq. (17), solving the block tri-diagonal system $AF_c = -B\tau^\delta$ is more time consuming and thus dominates the efficiency of the

CFA implementation, especially for systems with a large number of links. Several parallelization techniques exist for solving the tri-diagonal system, such as the *cyclic reduction algorithm* (CRA) and the RDA to name a few [28], [29]. As mentioned earlier, the parallel RDA is similar to the parallel scan operation used for accelerating the ID algorithm and partially the FD algorithms in our previous work [15]. However, RDA requires that all the strictly upper triangular blocks in the block tri-diagonal system be nonsingular, which in general does not hold for the CFA. Alternatively, we may resort to a variant of CRA, namely the OEEA, that does not depend on such assumption [30]. The same algorithm is adopted by Fijany in [14].

Intuitively, given a block tri-diagonal system with n diagonal blocks, the OEEA eliminates the off-diagonal blocks in $\lceil \log_2 n \rceil$ iterations ($\lceil\ \rceil$ denotes the ceiling function) until the original tri-diagonal matrix becomes block-diagonal. Fig. 1(d) illustrates the diagonalization process for a linear system with 8 diagonal blocks, which involves 3 eliminating-updating iterations. In step j ($1 \leq j \leq \lceil \log_2 8 \rceil$), the non-zero off-diagonal blocks (green) of the i-th row are eliminated by subtracting from them blocks of the $(i + 2^{j-1})$-th row or the $(i - 2^{j-1})$-th row multiplied by suitable coefficient matrices. The elimination process then updates all the diagonal blocks and some off-diagonal blocks (yellow) as shown in Fig. 1(d). These updated off-diagonal blocks will be annihilated in the next step in the same manner. Each elimination and updating calculation involves blocks from the last step and thus can be executed in parallel.

The constraint force matrix A in the CFA is a block tri-diagonal system with the following form:

$$A = \begin{bmatrix} D_1 & U_1 & & & \\ U_1^T & D_2 & U_2 & 0 & \\ & & \ddots & & \\ 0 & & U_{n-2}^T & D_{n-1} & U_{n-1} \\ & & & U_{n-1}^T & D_n \end{bmatrix} \tag{18}$$

Its diagonal blocks are all symmetric and invertible, and all the strictly lower triangular blocks are the transpose of the corresponding strictly upper triangular blocks. Therefore, we only need to update the upper triangular non-zero blocks as shown in Fig. 1(d). The right-hand side of the equation $AF_c = -B\tau^\delta$ is also updated in the same manner. For the j-th step, the update rule is as follows:

$$\begin{aligned} D_i^{(j)} &= D_i^{(j-1)} - E_i^{(j)}(U_i^{(j-1)})^T - K_i^{(j)} U_{i-2^{j-1}}^{(j-1)} \\ U_i^{(j)} &= -E_i^{(j)} U_{i+2^{j-1}}^{(j-1)} \\ R_i^{(j)} &= R_i^{(j-1)} - E_i^{(j)} X_{i+2^{j-1}}^{(j-1)} - K_i^{(j)} R_{i-2^{j-1}}^{(j-1)} \end{aligned} \tag{19}$$

where $D_i^{(j)}$, $U_i^{(j)}$ are the i-th diagonal and strictly upper triangular blocks in the matrix A and $R_i^{(j)}$ is the i-th block in $B\tau^\delta$. $E_i^{(j)} = U_i^{(j-1)}(D_{i+2^{j-1}}^{(j-1)})^{-1}$ is the coefficient matrix used to eliminate $U_i^{(j-1)}$ and $K_i^{(j)} = (U_{i-2^{j-1}}^{(j-1)})^T (D_{i-2^{j-1}}^{(j-1)})^{-1}$ is the coefficient matrix used to eliminate $(U_i^{(j-1)})^T$. From the fact that D_i is symmetric, we may compute the coefficient matrix

Algorithm 2: OEEASOLVBTDSYS($[D_i^{(0)}], [U_i^{(0)}], [R_i^{(0)}]$).

for $j \leftarrow 1$ **to** $\lceil \log_2 n \rceil$ **do**
  // Compute coefficients in parallel for $i = 1$ **to** $n - 2^{j-1}$.
  begin
    $(E_i^{(j)})^T \leftarrow$ SOLVSYS($[D_{i+2^{j-1}}^{(j-1)}], [(U_i^{(j-1)})^T]$)
    $(K_i^{(j)})^T \leftarrow$ SOLVSYS($[D_i^{(j-1)}], [U_i^{(j-1)}]$)
  end
  // Update D_i and R_i in parallel for $i = 1$ **to** $n - 2^{j-1}$.
  begin
    $D_i^{(j)} \leftarrow D_i^{(j-1)} - E_i^{(j)}(U_i^{(j-1)})^T$
    $D_{i+2^{j-1}}^{(j)} \leftarrow D_{i+2^{j-1}}^{(j)} - K_i^{(j)} U_i^{(j-1)}$
    $L_i^{(j)} \leftarrow E_i^{(j)} R_{i+2^{j-1}}^{(j-1)}$
    $R_{i+2^{j-1}}^{(j)} \leftarrow R_{i+2^{j-1}}^{(j-1)} - K_i^{(j)} R_i^{(j-1)}$
    $R_i^{(j)} \leftarrow R_i^{(j)} - L_i^{(j)}$
  end
  // Update U_i in parallel where $i = 1$ **to** $n - 2^j$.
  $U_i^{(j)} \leftarrow -E_i^{(j)} U_{i+2^{j-1}}^{(j-1)}$
end
// Compute all constraint forces in parallel where $j = \lceil \log_2 n \rceil$.
begin
  $[F_{i,c}] \leftarrow$ SOLVSYS($[D_i^{(j)}], [R_i^{(j)}]$)
end

$E_i^{(j)}$ by solving a less costly and more stable linear system $D_{i+2^{j-1}}^{(j-1)}(E_i^{(j)})^T = (U_i^{(j-1)})^T$, rather than directly taking the inverse of $D_{i+2^{j-1}}^{(j-1)}$. The other coefficient matrices $K_i^{(j)}$'s can be computed in the same manner. As shown in Fig. 1(d), $E_i^{(j)}$ and $D_i^{(j)}$ are only needed for $i = 1$ to $n - 2^{j-1}$, because the last 2^{j-1} strictly upper triangular blocks are removed in the j-th step. A more detailed description of this GPU-based parallel OEEA is shown in Algorithm 2.

V. EXPERIMENT

In this section, we compare the efficiency of the three parallel/hybrid FD algorithms in two different experiments for 3D serial robots with 1-DOF joints. We first investigate the scalability of each algorithm with increasing number of links for a single robot. Then, we demonstrate the efficiency of each method on a large group of robots with moderate number of links, which may pose a computation bottleneck in applications such as model-based control optimization. Before dynamics computations in all experiments, components describing the robot model, i.e. the joint twists, the initial adjacent link transformations and the links' inertia, are randomly constructed within reasonable range. While the parallel JSIIA and the parallel CFA are completely executed on a GPU, the sequential-parallel hybrid ABIA is implemented by iteratively computing the ABIs on CPU and executing the rest on GPU.

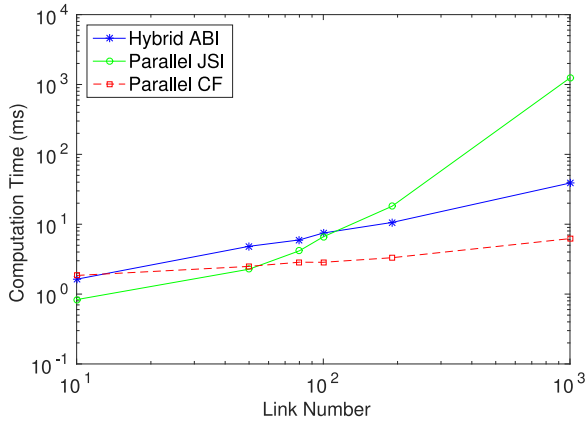

Fig. 2. Computation time comparison of different parallel forward dynamics algorithms for robots with different number of links.

Each experiment is repeated 1000 times with randomized joint inputs, after which the average computation time is reported. All experiments are conducted on a desktop workstation with an 8-core Genuine Intel i7-6700 CPU and 15.6 GB memory. We implement our GPU-based parallel dynamics algorithms using CUDA on a Tesla K40c GPU with a 11520 MB video memory and 288 GB/sec memory bandwidth.

A. Experiments With Different Numbers of Links

We compare the running time of a single FD call for robots with different numbers of links. As illustrated in Fig. 2, for a robot with 10 links, the parallel JSIIA outperforms the hybrid ABIA and the parallel CFA, with the latter two running at almost the same speed. However, as n increases substantially, the parallel CFA outperforms the hybrid ABIA, which in turn outperforms the parallel JSIIA. The relatively heavy overhead of the parallel CFA implementation with small n is expected because of the extra operations introduced to achieve the full parallelism. As the system scales up, such overhead gets mitigated by the high acceleration rate of the other parts within the parallel CFA. In comparison, the inversion of the JSI matrix (for JSIIA) and the computation of the ABIs via nonlinear recursion (for ABIA) pose a major bottleneck for the corresponding parallel/hybrid FD algorithms when n is large.

B. Experiments With Different Numbers of Groups

We now compare the efficiency of conducting a different numbers of groups of independent FD computations. Fig. 3(a) and (b) show respectively the performance of each algorithm for robots with 10 links and 200 links. For 10-link robots, the computational efficiency of the parallel JSIIA and the parallel CFA is similar when the number of groups is larger than 100; both are faster than the ABIA. The parallel JSIIA excels in this situation because the number of links is not large enough to reach its bottleneck (inversion of JSI). However, for 200-link robots, the parallel CFA always outperforms the other two algorithms, with the parallel JSIIA having the worst efficiency. The

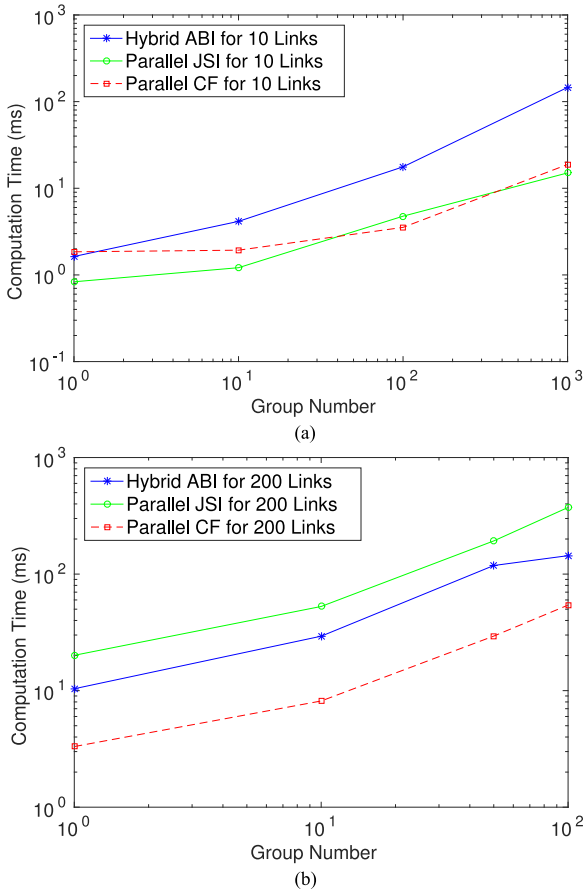

Fig. 3. Computation time comparison of different algorithms on many independent forward dynamics calls. (a) Comparison for robots with ten links. (b) Comparison for robots with 200 links.

efficiency degradation of the parallel JSIIA is clearly attributed to the JSI inversion.

In conclusion, the parallel CFA demonstrates the best performance for robots with a large number of links, while the parallel JSIIA is the most suitable for small scale systems. Due to the limited computational efficiency of sequential nonlinear recursive operations and overhead for data transfer between CPU and GPU, the performance of the hybrid ABIA is dominated by the parallel JSIIA and the parallel CFA.

VI. Discussion

So far, we have only considered FD algorithms for serial robot systems. Our future work shall extend the framework to cover more general robot topologies such as tree-structured and close-loop structured robot systems. Some relevant technical issues are discussed as follows. First, the implementation of the JSIIA is the same for tree-structures and closed-loop structures in the sense that once the JSI matrix is computed (by assembling branch JSIs or projecting onto the independent joint spaces), joint accelerations can be obtained by solving the same linear system as in Eq. (4). As pointed out in [15], the recursive ID algorithms for both serial and tree-structured systems can be accelerated by parallel scan, which in turn accelerates the

parallel scan-based FD algorithms. Some researchers have derived ID algorithms of a closed-loop systems from its corresponding reduced system [31], [32], i.e. a tree structure obtained by cutting loops, and mapping the computed torque vector of the reduced system to that of the original system. In this case, the scan-based ID and FD algorithms can be almost immediately applied to closed-loop systems, if the loop constraint Jacobian can be efficiently determined. On the other hand, the parallelization of the interbody-force algorithms such as the CFA hinges on the tri-diagonal structure of the constraint matrix, which essentially reflects the underlying serial kinematic topology. By applying the concept of ABI to branches, Featherstone and Fijany developed interbody-force based algorithms suitable for short-branched systems [27].

REFERENCES

[1] R. Featherstone, *Rigid Body Dynamics Algorithms*. New York, NY, USA: Springer, 2014.

[2] K. Ning and F. Wörgötter, "A novel concept for building a hyper-redundant chain robot," *IEEE Trans. Robot.*, vol. 25, no. 6, pp. 1237–1248, Dec. 2009.

[3] K. Yamane and Y. Nakamura, "Dynamics filter-concept and implementation of online motion generator for human figures," *IEEE Trans. Robot. Autom.*, vol. 19, no. 3, pp. 421–432, Jun. 2003.

[4] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2012, pp. 5026–5033.

[5] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *J. Dyn. Syst. Meas. Control*, vol. 104, no. 3, pp. 205–211, 1982.

[6] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *Int. J. Robot. Res.*, vol. 2, no. 1, pp. 13–30, 1983.

[7] G. Rodriguez, A. Jain, and K. Kreutz-Delgado, "A spatial operator algebra for manipulator modeling and control," *Int. J. Robot. Res.*, vol. 10, no. 4, pp. 371–381, 1991.

[8] I. Sharf and G. M. T. D'Eleuterio, "An iterative approach to multibody simulation dynamics suitable for parallel implementation," *J. Dyn. Syst., Meas.*, vol. 115, pp. 730–735, 1994.

[9] K. S. Anderson and S. Duan, "A hybrid parallelizable low-order algorithm for dynamics of multi-rigid-body systems: Part I, chain systems," *Math. Comput. Model.*, vol. 30, no. 9, pp. 193–215, 1999.

[10] R. Featherstone, "A divide-and-conquer articulated-body algorithm for parallel $O(\log(n))$ calculation of rigid-body dynamics. Part 2: Trees, loops, and accuracy," *Int. J. Robot. Res.*, vol. 18, no. 9, pp. 876–892, 1999.

[11] M. Poursina and K. S. Anderson, "An extended divide-and-conquer algorithm for a generalized class of multibody constraints," *Multibody Syst. Dyn.*, vol. 29, no. 3, pp. 235–254, 2013.

[12] A. Avello, J. M. Jiménez, E. Bayo, and J. G. de Jalón, "A simple and highly parallelizable method for real-time dynamic simulation based on velocity transformations," *Comput. Methods Appl. Mech. Eng.*, vol. 107, no. 3, pp. 313–339, 1993.

[13] J. Critchley and K. Anderson, "A parallel logarithmic order algorithm for general multibody system dynamics," *Multibody Syst. Dyn.*, vol. 12, no. 1, pp. 75–93, 2004.

[14] A. Fijany, I. Sharf, and G. M. D'Eleuterio, "Parallel $O(\log n)$ algorithms for computation of manipulator forward dynamics," *IEEE Trans. Robot. Autom.*, vol. 11, no. 3, pp. 389–400, Jun. 1995.

[15] Y. Yang, Y. Wu, and J. Pan, "Parallel dynamics computation using prefix sum operations," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1296–1303, Jul. 2017.

[16] L. Csanky, "Fast parallel matrix inversion algorithms," *SIAM J. Comput.*, vol. 5, no. 4, pp. 618–623, 1976.

[17] F. C. Park, J. E. Bobrow, and S. R. Ploen, "A Lie group formulation of robot dynamics," *Int. J. Robot. Res.*, vol. 14, no. 6, pp. 609–618, 1995.

[18] S. R. Ploen and F. C. Park, "Coordinate-invariant algorithms for robot dynamics," *IEEE Trans. Robot. Autom.*, vol. 15, no. 6, pp. 1130–1135, Dec. 1999.

[19] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge, U.K.: Cambridge University Press, 2017.

[20] K. Lee and G. S. Chirikjian, "A new perspective on $O(n)$ mass-matrix inversion for serial revolute manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2005, pp. 4722–4726.

[21] J. J. Laflin, K. S. Anderson, and M. Hans, "Enhancing the performance of the dca when forming and solving the equations of motion for multibody systems," in *Multibody Dynamics*. New York, NY, USA: Springer, 2016, pp. 19–31.

[22] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL, USA: CRC press, 1994.

[23] J. E. Marsden and T. Ratiu, *Introduction to Mechanics and Symmetry: A Basic Exposition of Classical Mechanical Systems*. New York, NY, USA: Springer, 2013.

[24] A. Jain, "Unified formulation of dynamics for serial rigid multibody systems," *J. Guid., Control, Dyn.*, vol. 14, no. 3, pp. 531–542, 1991.

[25] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, no. 8, pp. 786–793, Aug. 1973.

[26] G. E. Blelloch, "Prefix sums and their applications," Technical Report, School of Computer Science, Carnegie Mellon University, 1990.

[27] A. Fijany and R. Featherstone, "A new factorization of the mass matrix for optimal serial and parallel calculation of multibody dynamics," *Multibody Syst. Dyn.*, vol. 29, no. 2, pp. 169–187, 2013.

[28] S. Cohen, "Cyclic reduction," 1994. [Online]. Available: http://robotics.stanford.edu/ scohen/cr.pdf

[29] S. K. Seal, "An accelerated recursive doubling algorithm for block tridiagonal systems," in *Proc. IEEE 28th Int. Parall. Distrib. Process. Symp.*, 2014, pp. 1019–1028.

[30] C. Levit, "Parallel solution of pentadiagonal systems using generalized odd-even elimination," in *Proc. ACM/IEEE Conf. Supercomput.*, 1989, pp. 333–336.

[31] J. Wittenburg, *Dynamics of Systems of Rigid Bodies*, vol. 33. New York, NY, USA: Springer-Verlag, 2013.

[32] J. J. Murray and G. H. Lovell, "Dynamic modeling of closed-chain robotic manipulators and implications for trajectory control," *IEEE Trans. Robot. Autom.*, vol. 5, no. 4, pp. 522–528, Aug. 1989.