Parallel Dynamics Computation Using Prefix Sum Operations
Yajue Yang, Yuanqing Wu, and Jia Pan

Abstract—A new parallel framework for fast computation of inverse and forward dynamics of articulated robots based on prefix sums (scans) is proposed. We first re-investigate the well-known recursive Newton–Euler formulation for robot dynamics and show that the forward–backward propagation process for robot inverse dynamics is equivalent to two scan operations on certain semi-groups. Then, we showed that state-of-the-art forward dynamic algorithms can also be cast into a sequence of scan operations almost completely, with unscannable parts clearly identified. This suggests a serial–parallel hybrid approach for systems with a moderate number of links. We implement our scan-based algorithms on Nvidia CUDA platform with performance compared with multi-threading CPU-based recursive algorithms; a computational acceleration is demonstrated.

Index Terms—Direct/inverse dynamics formulation, prefix-sum operations, GPGPU.

I. INTRODUCTION

Merging applications of dynamics computation in humanoid robotics and model-based control optimization field involve simultaneously solving a large number of groups (>1000) of dynamics problems for a robot with a moderate number (10 ~ 100) of links [1]–[5]. This raises new computational challenges on classic articulated robot dynamics algorithms: hundreds to thousands of groups of inverse dynamics (ID) or forward dynamics (FD) and their derivatives under different states and control inputs must be evaluated within several milliseconds. Fortunately, such a computation task is inherently akin to parallel computation on multiple levels, and therefore may take substantially less time than purely sequential implementation in the perceived scenario. At the outset, data-independent parallelism from different states and inputs guarantees a parallel computation on the group level. Within individual group, data-dependent parallelism in ID and FD is well known to the robotics community, thanks to decades of extensive research on adapting recursive sequential algorithms to parallel algorithms for multi-processor systems [6]–[13]. A comprehensive summary of state-of-art sequential dynamics algorithms can be found in [14, ch. 5,6]. See also [15], [16].

Cherieten et al. have identified part of the ID as a prefix sum operation [4], also known as scan [17], which is useful and easy to implement building blocks for many parallel algorithms. However, they have not exploited the Lie group structure which provides a geometric and more intuitive perspective of robot dynamics. In fact, articulated robot kinematics and dynamics can be considered as a spatial propagation problem [18]–[20], which leads to linear recursions that are well known to be equivalent to scan operations [17]. This implies that the ID and FD problem may be mostly (if not entirely) formulated as a sequence of scan operations, thereby paving the way for applying state-of-the-art parallel computing technology to the classic dynamics computation problem without reinventing any wheels.

In this letter, we show that the parallel scan perspective on robot dynamics computation can indeed be carried to a further extent: apart from the initialization and some intermediate variables that can be computed in parallel, both the ID and the FD problem may be reformulated into a sequence of scan operations. The letter is organized as follows. In Section II, we give a brief review of the recursive Newton-Euler formulation for robot dynamics, following the Lie group approach formally introduced in [19]. Our main contribution is presented in Sections III and IV, where the recursive ID and problems are reformulated into a sequence of scan operations. We show that the scan operator can be considered as a binary operation on certain semigroups, and is therefore coordinate/representation invariant. Detailed algorithms of implementations are then described. In Section V, the experiment results of hybrid CPU/GPU implementations of our parallel scan-based algorithm and CPU-based multithreading implementations are compared. Finally, the scalability of our parallel computation framework for robots with open branched chains and other future works are discussed in Section VI.

II. LIE GROUP FORMULATION OF ROBOT DYNAMICS

For consistency with existing literature, we shall closely follow the notation used in [19], which is also summarized in Table I.
A. Inverse Dynamics

The ID problem of deriving necessary joint torques for generating a specified robot motion is well-known to be a forward-backward two-phase propagation process [19], [21], [22]. In the forward propagation phase, the motion of each joint (with twist axis $S_i$ and joint coordinates $q_i$ and their time derivatives $\dot{q}_i$, $\ddot{q}_i$) propagates from the base toward the end-effector (EE) of an $n$-link robot, resulting in a linear recursion of link velocities $V_i$ and accelerations $\ddot{V}_i$ for $i = 1, \ldots, n$:

$$
\begin{aligned}
&f_{i-1,i} = M_i e^{S_i q_i} \\
&V_i = \text{Ad}_{f_{i-1,i}}(V_{i-1}) + S_i \ddot{q}_i \\
&\ddot{V}_i = S_i \dddot{q}_i + \text{Ad}_{f_{i-1,i}}(\ddot{V}_{i-1}) - \text{ad}_{S_i} \ddot{q}_i \text{Ad}_{f_{i-1,i}}(V_{i-1}).
\end{aligned}
$$

(1)

In the back propagation phase, the reaction force applied through each joint to the succeeding link is successively computed using the Newton-Euler equation, which leads to a second set of linear recursion for joint reaction forces $F_i$ and joint torques $\tau_i$ for $i = n, \ldots, 1$:

$$
\begin{aligned}
&F_i = \text{Ad}_{f_{i-1,i}}^T(F_{i+1}) + J_i \dot{V}_i - \text{ad}_{\dot{q}_i}^T(J_i V_i) \\
&\tau_i = -\text{ad}^T_{\dot{q}_i} F_i. \\
\end{aligned}
$$

(2)

The base velocity and acceleration $V_0$, $\dot{V}_0$, and external force $F_{n+1}$ acting on the EE are provided to initiate the recursion. A sequential recursion of (1) and (2) leads to the $O(n)$ (for $n$ links) recursive Newton-Euler ID algorithm. Here, emphasis should be placed on the coordinate invariant Lie group description of robot kinematics and dynamics [21], [23]. We show in Section III that the scan operands and operators defined by (1) and (2) may be considered as the elements and binary operator of certain matrix semigroup that is closely related to $\text{SE}(3)$ (special Euclidean group of $\mathbb{R}^3$). This makes it possible to adapt our parallel framework to using other mathematical representations of $\text{SE}(3)$, such as dual quaternions [24] or other Clifford algebras [25]. For the rest of the letter, we shall denote the ID function by:

$$
\tau = \text{ID}(q, \dot{q}, \ddot{q}, V_0, \dot{V}_0, F_{n+1}).
$$

(3)

B. Forward Dynamics

The robot FD problem refers to the derivation of joint acceleration $\ddot{q} = (\ddot{q}_1, \ldots, \ddot{q}_n)^T$ as a function of the robot state $(\dot{q}^T, \dot{q}^T)^T$, applied joint torque $\tau$, base velocity and acceleration $V_0, \dot{V}_0$, and external force $F_{n+1}$:

$$
\ddot{q} = \text{FD}(q, \dot{q}, \tau, V_0, \dot{V}_0, F_{n+1}).
$$

(4)

It may be numerically integrated to compute joint position and velocity trajectories. The joint torques $\tau$ is linearly related to the joint accelerations $\ddot{q}$ via the joint space inertia (JSI) $M(q)$ with a bias term $\tau_{\text{bias}}$ accounting for Coriolis, centrifugal and external forces [14]:

$$
\begin{aligned}
&\tau = M(q) \ddot{q} + \tau_{\text{bias}} \\
&\tau_{\text{bias}} := \text{ID}(q, \dot{q}, \ddot{q} = 0, V_0, \dot{V}_0, F_{n+1})
\end{aligned}
$$

(5)

Since both the joint accelerations $\ddot{q}_i$’s and joint reaction forces $F_i$’s are unknown, the original FD problem obstructs a propagation formulation and cannot be directly solved by parallel scan operations. Alternatively, $\ddot{q}$ may be derived from directly inverting the JSI via

$$
\ddot{q} = M^{-1}(q) \dddot{q} := M^{-1}(q)(\tau - \tau_{\text{bias}}),
$$

(6)

hence leading to the $O(n^3)$ JSI inversion algorithm (JSIIA) [14, ch. 6], [26]. Depending on the methods for computing JSI, JSIIA may be computationally appealing when $n \leq 200$ as indicated in the following experiment results. However, it does not take advantage of the structure of JSI inherited from robot chain topology. This was emphasized in a series of letters on analytical factorization and the inverse of JSI [18], [27], [28]. The goal of the factorization is to represent the inverse of JSI as the product of block diagonal and block upper/lower triangular matrices, or a sequence of linear recursions that leads to the computationally efficient propagation methods [14, ch. 7].

The articulated-body inertia algorithm (ABIA) proposed by Featherstone [29] is a well-known propagation method that involves a nonlinear recursion for deriving the equivalent inertias of the sub-system rooted at link $i$ (see [23] for the notations) for

### Table I: Nomenclature

| Symbol | Description |
|--------|-------------|
| $\text{SE}(3)$ | Euclidean group of $\mathbb{R}^3$. |
| $f_{i-1,i}$ | Coordinate transformation from link frame $i-1$ to link frame $i$. |
| $\text{Ad}_{T_i}$ | $i$th joint twist. |
| $q_i$ | $i$th joint variable. |
| $V_i$ | Spatial velocity of $i$th link. |
| $\dot{V}_i$ | Spatial acceleration of $i$th link. |
| $F_i$ | Reaction force of $i$th joint. |
| $\tau_i$ | $i$th joint torque. |
| $J_i$ | $i$th link inertia. |
| $\hat{J}_i$ | $i$th articulated body inertia. |
| $M(q)$ | Joint space inertia at $q$. |
| $\text{GL}_n(n)$ | $n$th-order general linear group. |
| $[x_i]$ | Array $[x_1, \ldots, x_n]$. |
| $n\ell(3)$ | Lie algebra of $\text{SE}(3)$. |
| $M_0$ | Initial coordinate transformation of $f_{i-1,i}$. |
| $\text{ad}$ | Adjoint map. |
| $S$ | diag($S_1, \ldots, S_n$). |
| $q$ | Column $(q_1, \ldots, q_n)^T$. |
| $V$ | Column $(V_1, \ldots, V_n)$. |
| $F$ | Column $(F_1, \ldots, F_n)$. |
| $\tau$ | Column $(\tau_1, \ldots, \tau_n)^T$. |
| $J$ | diag$(J_1, \ldots, J_n)$. |
| $n\ell$ | $n$th-order matrix space. |
| $\oplus$ | Binary associative operator. |
\[ i = n, \ldots, 1: \]
\[
\hat{j}_i = j_{i+1} \text{Ad}_{\hat{f}_{i+1}^{-1}} \hat{j}_{i+1} \text{Ad}_{f_{i+1}^{-1}}
\]
\[
= \text{Ad}_{f_{i+1}^{-1}} \hat{j}_{i+1} S_{i+1} \text{Ad}_{f_{i+1}^{-1}} \hat{j}_{i+1} S_{i+1} \text{Ad}_{f_{i+1}^{-1}}
\]
\[
\frac{\hat{j}_{i+1} S_{i+1} \text{Ad}_{f_{i+1}^{-1}} \hat{j}_{i+1} S_{i+1} \text{Ad}_{f_{i+1}^{-1}}}{S_{i+1} \hat{j}_{i+1} S_{i+1} \text{Ad}_{f_{i+1}^{-1}}}
\]

which is essentially a recursive elimination of the unknown joint reaction forces \(F_i\)’s using virtual work principle. The availability of ABI allows for deriving link accelerations from joint torques \(\tau_i\)’s with a backward-forward two-phase propagation process [14], which may be summarized as follows [23]:

**Backward recursion:** for \(i = n, \ldots, 1\)
\[
\hat{\tau}_i = Y_{i,i+1} \hat{\tau}_{i+1} + \Pi_{i,i+1} \hat{\tau}_{i+1}
\]
\[
\hat{\lambda}_i = Y_{i,i+1} \hat{\lambda}_{i+1} + S_i \hat{c}_i
\]
\[
\hat{q}_i = \hat{c}_i - \Pi_{i,i+1} \hat{\lambda}_{i+1}
\]

**Forward recursion:** for \(i = 1, \ldots, n\)
\[
\hat{j}_i = Y_{i+1,i} \hat{j}_{i+1} + S_i \hat{e}_i
\]
\[
\hat{\lambda}_i = Y_{i+1,i} \hat{\lambda}_{i+1} + S_i \hat{e}_i
\]
\[
\hat{q}_i = \hat{e}_i - \Pi_{i+1,i} \hat{\lambda}_{i+1}
\]

where
\[
Y_{i,i+1} := \text{Ad}_{f_{i+1}^{-1}} \left( I - \frac{j_{i+1} S_{i+1} \text{Ad}_{f_{i+1}^{-1}}}{S_{i+1} j_{i+1} S_{i+1} \text{Ad}_{f_{i+1}^{-1}}} \right)
\]
\[
\Pi_{i,i+1} := \frac{\text{Ad}_{f_{i+1}^{-1}} \hat{j}_{i+1} S_{i+1} \text{Ad}_{f_{i+1}^{-1}}}{S_{i+1} \hat{j}_{i+1} S_{i+1} \text{Ad}_{f_{i+1}^{-1}}}
\]

The connection of ABI to square factorization of JSI is emphasized in [18]. All recursions involved in ABIA except that of (7) are linear and may eventually be formulated as scan operations. However, (7) is a well-known nonlinear recursion that does not speed beyond a constant factor by parallel algorithms [30], and therefore should not conform to a parallel scan operation. The joint reaction forces may also be eliminated by defining appropriate (not necessarily orthogonal) complements for joint constraint force, hence leading to the constraint force algorithm (CFA) [31].

### III. PREFIX-SUM RE-FORMULATION OF ROBOT DYNAMICS

**A. A Brief Review of Scan Operation**

The *all-prefix-sums* (scan) operation [17] takes a binary associative operator \(\oplus\), and an array of \(n\) elements
\[
[ a_0, a_1, \ldots, a_{n-1} ]
\]
and returns the array
\[
[ x_0, x_1, \ldots, x_{n-1} ] := [ a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1}) ].
\]
It is obvious that the scan operation implies the following recursion:
\[
x_i = x_{i-1} \oplus a_i.
\]

Generalizations of the standard scan operation and their applications are discussed in [17]. The parallel scan algorithm regards the operands as leaves of a balanced tree. The operation may then be summarized as: first sweeping up the tree to generate partial sums at internal nodes, followed by a down sweep to obtain the array of all-prefix sums.

**B. Synchronous Forward Scan of LinkVelocities and Accelerations**

To derive the forward scan in the ID algorithm, we start by rewriting the second and third equation of (1) in a single linear recursion:
\[
\begin{bmatrix}
\dot{V}_i \\
\ddot{V}_i \\
V_{i-1}
\end{bmatrix} =
\begin{bmatrix}
\text{Ad}_{f_{i-1}^{-1}} & -\text{Ad}_{f_{i-1}^{-1}} & S_i \hat{q}_i \\
0 & \text{Ad}_{f_{i-1}^{-1}} & S_i \hat{q}_i \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\dot{V}_{i-1} \\
\ddot{V}_{i-1} \\
V_{i-1}
\end{bmatrix}
\]

leading to a synchronous scan of both link velocities \(V_i\)’s and accelerations \(\ddot{V}_i\)’s. Emphasis should be made on the Lie group structure of the operands: it can be shown that the operands are isomorphic to \((f_{i-1}^{-1} S_i \hat{q}_i, S_i \hat{q}_i) \in \text{SE}(3) \times \text{se}(3)^2\), a product group with binary operation \(\oplus\) defined by:
\[
(g, \xi_1, \xi_2) \oplus (g', \xi'_1, \xi'_2) :=
\]
\[
\left( gg', \text{Ad}_g(\xi'_1) + \xi_1 - \text{Ad}_g(\xi_1), \text{Ad}_g(\xi'_2) + \xi_2 \right).
\]

The identity element of this Lie group is \((I, 0, 0)\), and the inverse of \((g, \xi_1, \xi_2) \in \text{SE}(3) \times \text{se}(3)^2\) is given by:
\[
(g^{-1}, -\text{Ad}_{g^{-1}}(\xi_1), -\text{Ad}_{g^{-1}}(\xi_2)).
\]

We may take advantage of this isomorphism by first scanning the isomorphic operands \((f_{i-1}^{-1} S_i \hat{q}_i, S_i \hat{q}_i)\)’s and then performing a parallel isomorphism to recover \(V_i\)’s and \(\ddot{V}_i\)’s.

**C. Backward Scan of Reaction Force and Joint Torques for Inverse Dynamics**

After scanning the link velocities and accelerations, the bias term \(F_i := J_i V_i - \text{ad}_{\hat{q}_i} (J_i V_i)\) in the first equation of (2) may be pre-computed in parallel before a second scan computes the linear recursion for \(F_i\)’s. Alternatively, \(F_i\) is quadratic in \(V_i\) and linear in \(\dot{V}_i\), and may be synchronously computed along with the scan of (15). Moreover, it can be verified that \(F_i\) involves only 9 out of the 21 quadratic terms of \(V_i = (v_1^T, w_1^T)^T\), namely \(w_j w_k \leq i \leq k \leq 3\) (where \(w_i = (w_{11}, w_{12}, w_{13})^T\)), and \(w_i \times v_j \in \mathbb{R}^3\). For convenience, we define \(Q_i = ((w_j \times v_i)^T, w_j^T, w_k^T, w_{11} w_{12}, w_{11} w_{13}, w_{12} w_{13}, w_{12} w_{13}, w_{12} w_{13})^T \in \mathbb{R}^9\).
synchronous scan may be expressed as follows:

\[
\begin{bmatrix}
\dot{V}_1 \\
Q_1 \\
V_1 \\
F_1 \\
1
\end{bmatrix} =
\begin{bmatrix}
* & 0 & 0 & * \\
0 & * & 0 & 0 \\
0 & 0 & * & 0 \\
* & 0 & 0 & * \\
0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\dot{V}_{i-1} \\
Q_{i-1} \\
V_{i-1} \\
F_{i-1} \\
1
\end{bmatrix}
A_i \in \mathbb{R}^{28 \times 28}
\]

where only the block pattern of the scan operand \( A_i \) is shown. A similar simplification of the operand by isomorphism may be further investigated. After scanning the forward propagation (18), we proceed with the scan of the backward propagation (2) using the following linear recursion:

\[
\begin{bmatrix}
F_1 \\
\tau_{i+1} \\
1
\end{bmatrix} =
\begin{bmatrix}
\text{Ad}_T \mathbf{f}_{i+1} \\
S_{i+1} \\
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
F_{i+1} \\
\tau_{i+2} \\
1
\end{bmatrix}
A_i \in \mathbb{R}^{8 \times 8}
\]

\[
F_0 = \tau_{n+1} = \tau_{n+2} = 0.
\]

We have hitherto shown that the ID problem may be completely solved by two scanning operations.

\[18\]

\[19\]

D. Accelerating JSIIA Using Scan Operations

Following the discussion about (6) in Section II-B, we may accelerate the JSIIA by parallelizing data-independent computation of the columns \( M_{..j}(q) \) of the JSI \( M(q) \) (see [14, Ch. 6] for more details) using our parallel ID algorithm:

\[M_{..j}(q) = M(q)\delta_{.j} = \text{ID}(q, \dot{q}, \ddot{q} = \delta_{.j}, V_0, \dot{V}_0, F_{n+1}) - \text{ID}(q, \dot{q}, \ddot{q} = 0, V_0, \dot{V}_0, F_{n+1}) = \text{ID}(q, 0, \delta_{.j}, 0, 0, 0)\]

\[\delta_{.j} := (\delta_{1,j}, \ldots, \delta_{n,j})^T \quad j = 1, \ldots, n \tag{20}\]

where \( \delta_{1,j} \) denotes the Kronecker delta function. The joint accelerations \( \dot{q} \) may then be evaluated from (6) using state-of-the-art parallel linear system solvers. Consequently, a total of \( n+1 \) parallel IDs (including the one for computing \( \tau_{\text{bias}} \)) are computed in parallel for the JSIIA.

E. Accelerating ABIA Using Scan Operations

The details of square factorization of JSI and its analytical inversion may be found in [23]. Following our discussion in Section II-B, the two-phase propagation summarized in (9) may be reformulated into a backforward scan:

\[
\begin{bmatrix}
\dot{v}_1 \\
\dot{c}_{i+1} \\
1
\end{bmatrix} =
\begin{bmatrix}
Y_{i+1} T, i \\
\Omega_{i+1} S_{i+1} \\
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
\dot{v}_{i+1} \\
\dot{c}_{i+1} \\
1
\end{bmatrix}
A_i \in \mathbb{R}^{8 \times 8}
\]

\[
\dot{v}_{n+1} = \dot{c}_{n+1} = \dot{c}_{n+2} = 0
\]

followed by a forward scan:

\[
\begin{bmatrix}
\lambda_1 \\
\lambda_{i-1} \\
1
\end{bmatrix} =
\begin{bmatrix}
Y_{i-1} T, i \\
-\Omega_{i-1} S_{i-1} \\
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
\lambda_{i-1} \\
\lambda_i \\
1
\end{bmatrix}
A_i \in \mathbb{R}^{8 \times 8}
\]

\[\lambda_0 = \dot{q}_0 = 0 \tag{22}\]

respectively. Therefore, the complete ABIA comprises two scan operations on operands \( A_i \)'s given in (18), (19) for the computation of bias torques \( \tau_{\text{bias}} \)'s, a nonlinear recursion (7) for the computation of the ABI, and then finally two scan operations on operands \( A_i \)'s given in (21), (22) for the computation of joint accelerations \( \dot{q}_i \)'s. We may also combine the two backward scans pertaining to (19) and (21) into a single backward scan, with the corresponding linear recursion given by:

\[
\begin{bmatrix}
\dot{v}_{i-1} \\
\dot{c}_{i+1} \\
1
\end{bmatrix} =
\begin{bmatrix}
\text{Ad}_T \mathbf{f}_{i-1} \\
-\mathbf{S}_{i} \\
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
\dot{v}_{i-1} \\
\dot{c}_{i+1} \\
1
\end{bmatrix}
A_i \in \mathbb{R}^{15 \times 15}
\times
\begin{bmatrix}
F_i \\
\dot{v}_{i+1} \\
\dot{c}_{i+1} \\
1
\end{bmatrix}
\tag{23}\]

thereby reducing the hybrid ABIA to one nonlinear recursion to be executed on the CPU, and a sequence of three parallel scans to be executed on the GPU.

IV. IMPLEMENTATION

So far, we have presented a complete parallel framework for utilizing parallel scans to accelerate articulated robot ID and FD computation. In comparison to the classic recursive algorithms [23], we have eliminated a large portion of intermediate variables by assembling multiple linear recursions to a minimal number of large dimension linear recursions (two for ID, and three for ABIA). However, such a mathematically compact form does not necessarily lead to an optimal data structure for implementation on a particular hardware platform. In this section, we shall materialize the corresponding parallel algorithms on a hybrid CPU-GPU platform, and illustrate how hardware constraints may weigh in to tailor our algorithms with the flexibility of composing/decomposing linear recursions. We adopt Nvidia
Algorithm 1: Parallel Inverse Dynamics \texttt{CALCINVDyn} ([M], [q]), [q]).

Parallel Compute: Compute adjacent transformations in parallel.
1: \{f_{i-1,i}\} \leftarrow \texttt{CALCTRANSFORM}([M], [S]), [q])

Forward Scan: Compute body velocities.
2: \{V_i\} \leftarrow \texttt{INCLUSIVEVELSCAN}([f_{i-1,i}], [S]), [q])

Forward Scan: Compute body accelerations.
3: \{V_i\} \leftarrow \texttt{INCLUSIVEACCCSCAN}([f_{i-1,i}], [S]), [q], [V])

Backward Scan: Compute body forces.
4: \{F_i\} \leftarrow \texttt{INCLUSIVEFORCESCAN}([f_{i-1,i}], [V], [V])

Parallel Compute: Compute joint torques.
5: [\tau_i] \leftarrow \texttt{CALCTORQUE}([S], [F])

CUDA as our software to make our implementation easily reproducible for further study and comparison.

A. Parallel Inverse Dynamics Algorithm

In Section II-A, we proposed a forward-backward double scan parallelization of the recursive ID algorithm [19], where the forward scan computes all link velocities and accelerations (along with the bias force) and the backward scan calculates the generalized force transmitted through joints. In the actual GPU implementation, we split the scan for velocities and accelerations into two successive scans so that the scan data may be fitted into the GPU’s high-speed shared memory and local registers. The computation of bias forces \(F_i\)’s becomes perfectly parallel after the velocity and acceleration scans, and may simply be computed on \(n\) GPU threads. The details of our parallel ID algorithm are illustrated in Algorithm 1.

B. Forward Dynamics Algorithm

As shown in Section III, the parallelization of FD algorithms such as JSIIA and \(\text{ABI}\)A relies on recognizing the parallel ID algorithm as a common subroutine to compute the bias torques and/or the inertia matrix. Although, neither FD algorithms are completely scannable, the JSIIA may be alternatively accelerated by parallelly solving a positive definite linear system of equations. The \(\text{ABI}\), on the other hand, relies on CPU recursion to compute the ABI, and may be accelerated by careful scheduling of the hybrid CPU/GPU system.

1) Parallel JSIIA: Given a robot with \(n\) links, JSIIA calls the ID routine \(n + 1\) times with different inputs. The first call computes the bias torques \(\tau_{bias}\)’s by setting zero-acceleration input, i.e., \(\dot{q}_i = 0, i = 1, \ldots, n\). The other \(n\) ID calls compute the \(n\) columns of the JSI \(M(q)\) by setting \(\dot{q} = \delta_{.,j}, j = 1, \ldots, n\). All \(n + 1\) ID calls are data independent, and may be solved simultaneously on GPUs. Besides, the additional matrix operations involved in the JSIIA can also be computed in parallel on GPUs for further acceleration. The details of our parallel JSIIA are illustrated in Algorithm 2.

2) Hybrid \(\text{ABI}\): In comparison to JSIIA, \(\text{ABI}\) only leverages the parallel ID routine once for bias torque computation. The \(\text{ABI}\) \(J_i\)’s are recursively computed using (7) on the CPU and sent back to the GPUs for subsequent FD computation. The high latency of sending data from CPU main memory to the GPU video memory is partially hidden via asynchronously computing \(\text{ABIs}\). All remaining components can be parallelized on GPUs, with data transfer only occurring among device memories. This includes the computation for several intermediate variables, which are either perfectly parallel (line 4, 6, 8 in Algorithm 3) or scannable (line 5, 7 in Algorithm 3). The complete description of the hybrid \(\text{ABI}\) is shown in Algorithm 3.

V. EXPERIMENTS

In this section, we evaluate the performance of our GPU-based parallel dynamics algorithms by comparing their running time to that of their multithreading CPU counterparts on two experiments. The first experiment investigates the speedup and scalability with respect to the number of links \(n\) of a single
robot. The second one demonstrates efficacy on a large group of robots with small \( n \), which is a bottleneck scenario in many applications such as model-based control optimization. The robot configuration parameters such as twists are randomly generated before the dynamics computation in all our experiments.

Each experiment is repeated 1000 times with random joint inputs, with the average computation time reported. All running times are recorded on a desktop workstation with an 8-core Genuine Intel i7-6700 CPU and 15.6 GB memory. We implemented 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. CPU multithreading is implemented to parallelize the large number of independent dynamics computations in the second set of experiments, where the number of threads is equal to that of independent dynamics computations, while the calculations within individual group are still sequential. The results of the comparison are highly dependent on hardware and implementation. In our comparison, we have not applied any code optimization for the CPU implementation, which may have resulted in a decreased performance [32].

A. Experiments With Different Number of Links

We first compare the GPU and CPU’s performance of computing a single ID for robots as \( n \) changes, and the result is shown in Fig. 1(a). We observe a linear increase of the serial CPU’s running time with increasing \( n \), in comparison to a \( \log(n) \) increase of that of the GPU.

Next, we compare the time cost of a single FD call for robots with different \( n \). We investigate four different implementations of the FD computation, including GPU parallel JSIIA, CPU-GPU hybrid ABIA, serial JSIIA and serial ABIA. According to the result as shown in Fig. 1(b), the performance of serial JSIIA is dominated by other three methods. The GPU-based parallel JSIIA is the fastest for robots when \( n \) is small, but it is outperformed by hybrid ABIA and the serial ABIA when \( n \geq 100 \) and \( n \geq 190 \) respectively. This is because the JSIIA involves the inversion of inertia matrix. For robots with large \( n \), the inertia matrix inversion is more computationally expensive than any components in the ABIA, and may also involve expensive data exchange between global and local memory. Therefore, our hybrid ABIA is more efficient with robots having more than 100 links, thanks to the asynchronous data exchange which hides the latency of sending data from CPUs to GPUs.

B. Experiments With Different Number of Groups

We now compare the performance of carrying out a large number of independent dynamics computations between GPU-based parallel algorithms and multithreading CPU algorithms. In the CPU implementation, we allocate one thread for each independent dynamics computation.

The comparison result for ID implementations are shown in Fig. 2(a) and (b) for robots with 10 and 100 links respectively. We see that the running time of the CPU ID increases linearly with growing number of groups. In contrast, the computation time of the GPU ID increases much slower. Due to limited memory and necessary thread management of the GPU, the GPU ID algorithm does not reach the ideal \( O(\log(n)) \) time complexity. Nevertheless, it provides a satisfactory \( \sim 100\times \) speedup when the number of groups is 1000.

Next, four different approaches for computing the FD are compared, namely GPU-based parallel JSIIA and ABIA, CPU multithreading JSIIA and ABIA. According to the results as shown in Fig. 3(a), the GPU-based JSIIA is always the most efficient one. When \( n = 200 \), a different result is observed, as shown in Fig. 3(b): the hybrid ABIA always outperforms the other three methods, and the multithreading JSIIA is the slowest approach. This phenomenon is consistent with the result shown in Fig. 1(b) where the GPU-based JSIIA and the hybrid ABIA have the shortest running time when \( n = 10 \) and \( n = 200 \) respectively. This may be explained by the fact that when \( n \) is small (e.g., 10), the inertia matrix is small enough to be fit in the high-speed memory of the GPUs for high performance; we can further accelerate the dynamics computation by parallelizing matrix inversions in independent dynamics calls. When \( n \) is large (e.g., 200), the dimension of the inertia matrix becomes significantly high so that the running time for a single matrix inversion is far more computationally expensive than other
components in the dynamics computation. Worse still, due to limited GPU memory, we have to perform these time-consuming matrix inversions sequentially and consequently fail to leverage GPUs’ parallel mechanism.

VI. DISCUSSION AND FUTURE WORK

The parallel framework we have presented for computing ID and FD using parallel scan operations may also be generalized to robots with (rooted) tree structure (with the root identified as the base) if the base velocity and acceleration are given. In particular, its ID may be computed by recursive Newton-Euler algorithm similar to that of a serial robot [14]. We shall briefly discuss its formulation in this section.

In reference to (2), the joint reaction forces for the tree system are given by:

\[ F_i = \sum_{j \in \mu(i)} \text{Ad} F_{j,i} (F_j) + F_i^n \]  

(24)

where \( \mu(i) \) denotes the index set of all child links of link \( i \) and \( F_i^n = J_i V_i - \text{ad} F_{i,i} (J_i V_i) \). Since there exists only one path connecting the base and each EE, as shown in Fig. 4(b), the reaction forces contributed by each path to link \( i \) can be computed by (2):

\[ F^{p_j}_i = \text{Ad} F_{j,i} (F^{p_j}_{c(i,p_j)}) + F_i^n \]  

(25)

where \( c(i,p_j) \) denotes the index of link \( i \)'s child on path \( p_j \). We have \( F_i = \sum_{j \in \lambda(i)} F^{p_j}_i - (|\lambda(i)| - 1) F_i^n \) if all subtrees starting at the children of link \( i \) are serial. Here, \( \lambda(i) \) denotes the index
set of all paths that pass through link \( i \), and \( |\lambda(i)| \) denotes its cardinality. Finally, it may be proved by induction that:

\[
F_i = \sum_{j \in \lambda(i)} F_{i^j} - (|\lambda(i)| - 1) F_{i^0} - \sum_{q \notin \lambda(i)} (|\lambda(q)| - 1) \text{Ad}^T_{F_{i^q} \rightarrow F_{i^q}} (F_{i^q})
\]

(26)

where \( \nu(i) \) denotes the index set of all links of the subtree starting at link \( i \).

To summarize, the ID of a tree-structured robot can be processed by first computing in parallel the ID of each path using our parallel computation framework and then removing repetitive terms. Further details of this approach do not fit in this letter and will be picked up in our future work.

For a robot with unknown base velocity and acceleration, the ID problem is essentially hybrid. Consequently, it has to be solved by a hybrid dynamics algorithm [14]. The exploitation of our parallelization algorithms in general floating-base applications is part of our ongoing research. Nevertheless, our parallel dynamics algorithm can already be directly implemented in several special cases when the base velocity and accelerations can be calculated in advance [33] or when the base is fixed [34].

We shall also expand our parallel framework to include parallel scan of gradient and Hessian of ID [2], and other parallel FD algorithms such as CFA [31], and also an automatic scheduler for hardware-specific optimal performance. Besides, since our parallel framework is based on a coordinate-free Lie group/semigroup formulation and is not representation-specific, parallel algorithms using other representations, such as Clifford algebra [25] and geometric algebra [35], will be explored to further improve its performance.

REFERENCES

[1] 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.

[2] S.-H. Lee, J. Kim, F. C. Park, M. Kim, and J. E. Bobrow, “Newton-type algorithms for dynamics-based robot movement optimization,” IEEE Trans. Robot., vol. 21, no. 4, pp. 657–667, Aug. 2005.

[3] S. Lengagne, J. Vaillant, E. Yoshida, and A. Kheddar, “Generation of whole-body optimal dynamic multi-contact motions,” Int. J. Robot. Res., vol. 32, no. 9–10, pp. 1104–1119, 2013.

[4] B. Chretien, A. Escande, and A. Kheddar, “GPU robot motion planning using semi-infinite nonlinear programming,” IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 10, pp. 2926–2939, Oct. 2016.

[5] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control,” in Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst., 2012, pp. 5026–5033.

[6] 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.

[7] S. Chung and E. J. Haug, “Real-time simulation of multibody dynamics on shared memory multiprocessors,” J. Dyn. Syst., Meas. Control, vol. 115, no. 4, pp. 627–637, 1993.

[8] R. Hwang, D. Bae, J. Kuhl, and E. Haug, “Parallel processing for real-time dynamic system simulation,” J. Mech. Des., vol. 112, no. 4, pp. 520–528, 1990.

[9] H. Kasahara, H. Fujii, and M. Iwata, “Parallel processing of robot motion simulation,” in Proc. Triennial World Congr. Int. Fed. Autom. Control., vol. 4, 1987, pp. 329–336.

[10] R. H. Lathrop, “Parallelism in manipulator dynamics,” Int. J. Robot. Res., vol. 4, no. 2, pp. 80–102, 1985.

[11] L. Mráz and M. Valášek, “Solution of three key problems for massive parallelization of multibody dynamics,” Multibody Syst. Dyn., vol. 29, no. 1, pp. 21–39, 2013.

[12] R. Featherstone, “A divide-and-conquer articulated-body algorithm for parallel \( O((\log n)^3) \) calculation of rigid-body dynamics—Part 2: Trees, loops, and accuracy,” Int. J. Robot. Res., vol. 18, no. 9, pp. 876–892, 1999.

[13] K. S. Anderson and S. Duan, “Highly parallelizable low-order dynamics simulation algorithm for multi-rigid-body systems,” J. Guid., Control Dyn., vol. 23, no. 2, pp. 355–364, 2000.

[14] R. Featherstone, Rigid Body Dynamics Algorithms. Berlin, Germany: Springer, 2014.

[15] P. E. Nielsen, “Efficient computer simulation of motions of multibody systems.” Dissertation Abstr. Int. Part B: Sci. Eng., vol. 47, no. 12, 1987.

[16] K. S. Anderson, “Recursive derivation of explicit equations of motion for efficient dynamic/control simulation of large multibody systems,” 1990.

[17] G. E. Blelloch, “Prefix sums and their applications,” 1990.

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

[19] 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.

[20] A. Müller and P. Mässer, “A Lie-group formulation of kinematics and dynamics of constrained MBS and its application to analytical mechanics,” Multibody Syst. Dyn., vol. 9, no. 4, pp. 311–352, 2003.

[21] 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.

[22] J. Y. Luh, M. W. Walker, and R. P. Paul, “On-line computational scheme for mechanical manipulators,” J. Dyn. Syst., Meas. Control, vol. 102, no. 2, pp. 69–76, 1980.

[23] 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.

[24] J. R. Dooley and J. M. McCarthy, “Spatial rigid body dynamics using dual quaternion components,” in Proc. IEEE Int. Conf. Robot. Autom., vol. 1, Apr. 1991, pp. 90–95.

[25] J. Selig and E. Bayo-Corochano, “Rigid body dynamics using Clifford algebra,” Adv. Appl. Clifford Algebras, vol. 20, no. 1, pp. 141–154, 2010.

[26] 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.

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

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

[30] L. Hyafil and H.-T. Kung, “The complexity of parallel evaluation of linear recurrences,” J. ACM, vol. 24, no. 3, pp. 513–521, 1977.

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

[32] V. W. Lee et al., “Debunking the 100x GPU vs. CPU myth: An evaluation of throughput computing on GPU and CPU,” ACM SIGARCH Comput. Archit. News, vol. 38, no. 3, pp. 451–460, 2010.

[33] Q. Huang et al., “Planning walking patterns for a biped robot,” IEEE Trans. Robot. Autom., vol. 17, no. 3, pp. 280–289, Jun. 2001.

[34] S. Lengagne, N. Ramdani, and P. Fraisse, “Safe motion planning computation for balancing balanced movement of humanoid robots,” in Proc. IEEE Int. Conf. Robot. Autom., 2009, pp. 1669–1674.

[35] L. Dorst, L. Dorst, and F. Fontijne, Geometric Algebra for Computer Science: An Object-oriented Approach to Geometry. San Mateo, CA, USA: Morgan Kaufmann, 2009.