Dadu: Accelerating Inverse Kinematics for High-DOF Robots

Shiqi Lian[†‡], Yinhe Han[†], Ying Wang[†], Yungang Bao[†], Hang Xiao[†‡], Xiaowei Li[†] and Ninghui Sun[†]
[†]State Key Laboratory of Computer Architecture,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China
[‡]University of Chinese Academy of Sciences, Beijing, P.R. China
{lianshiqi, yinhes, wangying2009, baoyg, xiaohang, lxw, snh}@ict.ac.cn

ABSTRACT
Kinematics is the basis of robotic control, which manages the robots' movement, walking and balancing. As a critical part of Kinematics, the Inverse Kinematics (IK) will consume more time and energy to figure out the solution with the degrees of freedom increase. It goes beyond the ability of general-purpose processor based methods to provide real-time IK solver for manipulators with high degree of freedom. In this paper, we present a novel parallel algorithm, Quick-IK, based on the Jacobian transpose method. Via speculative searching in parallel, Quick-IK can reduce the number of iterations by 97% for the baseline Jacobian transpose method. In addition, we propose a novel specialized architecture, IKAcc, to boost the energy efficiency of Quick-IK through hardware acceleration. The evaluation shows that IKAcc can solve IK problem in 12 milliseconds for a 100 degrees of freedom manipulator. In addition, IKAcc can achieve 1700x performance speed-up over the CPU implementation of the original Jacobian transpose method and 30x speedup over the GPU implementation of Quick-IK. At same time, IKAcc achieves about 776x higher energy efficiency than the GPU implementation of Quick-IK.

1. INTRODUCTION
With the development of sensory technology and artificial intelligence, various advanced robots are built to perform a variety of tasks in our daily life. The mobile robot BigDog, built by Boston Dynamics, can climb and carry heavy load on rough terrain [1]. The manipulator KUKA KR AGILUS can play Ping-pong with the world champion Timo Boll [2]. For a robot or manipulator, its capability of dealing with complex tasks depends on the degree of freedom (DOF), e.g. NASA Valkyrie has 44 DOF [3]. With the increase of DOF, the computation complexity of inverse kinematics will rapidly grow and consume more time and energy. The inverse kinematics solver in ROS [13] in mobile computer will take over 1 second to obtain the results for 100 degrees of freedom manipulator, which cannot satisfy the criteria for real-time robotic control.

Robot kinematics contains forward kinematics (FK) and inverse kinematics (IK). As shown in Figure 1, the forward kinematics is to compute the position $\boldsymbol{X}$ of the end-effector from the joint angles $\boldsymbol{\theta}$. On the contrary, the goal of inverse kinematics is to compute the joint angles $\boldsymbol{\theta}$ from the end-effector's position $\boldsymbol{X}$.

DAC '17, June 18-22, 2017, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/3061639.3062223

The inverse kinematics is usually more complex than the forward kinematics. The forward kinematics can be solved through the kinematic equations directly, but the inverse kinematics may have multiple possible solutions or no solution at all. Many methods have been proposed to solve the inverse kinematics: Cyclic Coordinate Descent methods [4], Inverse Jacobian methods [5, 6, 7, 16, 19, 20] and Artificial Intelligence methods [8, 9, 10]. Compared with other methods, Inverse Jacobian methods are more generic and precise [11]. There are mainly two kind of Inverse Jacobian methods: the pseudoinverse method [5, 19, 20] and the transpose method [6, 7]. Usually, the pseudoinverse method takes a small number of iterations to converge, but will take a long time to calculate the pseudoinverse of matrix through singular value decomposition (SVD), which is hard to parallelize [12]. In contrast, the transpose method will take a large number of iterations to converge, but use the transpose of matrix instead of the pseudoinverse of matrix, which is easy to compute and parallelize. Conventionally speaking, the pseudoinverse method is considered faster than the transpose method in serial process due to less number of iterations. However, we found that the transpose method provides a larger space to speedup through multithreads architecture.

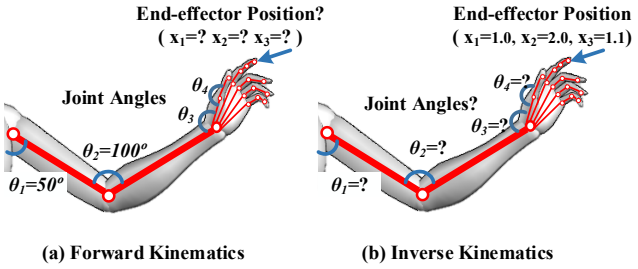


Figure 1. Robot Kinematics

In this paper, instead of resorting to the solutions that converges rapidly in general sense, we choose to exploit the potential in the transpose method and use it to build as a powerful real-time IK solver for high DOF kinematics. To accelerate the transpose method, we will adopt a novel approach to reduce the number of iterations, and explore the parallelism of each iteration through hardware. The key contributions of this work are:

- We propose a speculation-based approach, Quick-IK, to reduce the number of iterations in transpose method. In this approach, we speculate on multiple candidate parameter values in parallel and select the best parameter value to accelerate the convergence at each iteration.
- We propose an energy-efficient accelerator architecture for Quick-IK. The new design exploits the parallelism of each iteration by assigning each serially processed speculations into a fine-grained parallel process unit.

- We evaluate the proposed algorithm Quick-IK and the new architecture IKAcc. The evaluation result consists of (i) the efficiency of Quick-IK compared with the original transpose method and the pseudoinverse method, (ii) the performance and power comparisons between GPGPU solution (NVIDIA Jetson TX1) and IKAcc.

The rest of the paper is organized as follows. The related work is present in Section II. Section III describes the preliminaries of the Inverse Jacobian methods. Section IV presents the proposed optimal transpose method, Quick-IK. In Section V, we present a novel architecture for Quick-IK. Experimental results are presented in Section VI. Finally, conclusions and future work are presented in Section VII.

2. RELATED WORK

The inverse kinematics is a critical issue for robotic system. There are mainly three kinds of solvers: algebraic and geometric methods [4], learning methods [8, 9, 10], and numerical methods [5, 6, 17, 18]. The algebraic and geometric methods are just used in special manipulators, with finite and fixed solutions. The learning methods are easy to scale to any manipulators. But the learning methods have two challenges: training stage and accuracy. To use the neural network, a necessary step is to train the neural network, which depends on the training data and takes a long time to converge. While the numerical methods are used in common because they are fast and suitable for most of manipulators. In numerical methods, the inverse Jacobian methods are more stable than others.

There are several works that accelerate the inverse Jacobian methods, but the most of them are based on the pseudoinverse method. Olaru adopted neural network to compute the pseudoinverse of Jacobian Matrix $\boldsymbol{J}$, but the approximate $\boldsymbol{J}^{-1}$ may be worse than the transpose of $\boldsymbol{J}$ [19]. Buss adopted a selectively damped least squares to accelerate the convergence of the pseudoinverse method, but the improvement is limited [20].

Other than these methods in algorithm, there are some methods to solve IK through multithread architectures or special hardware efficiently. Kung proposed a FPGA-realization of inverse kinematics [21]. But the method is just suitable for two type low-DOF robots with 4-5 degrees of freedom. Hildenbrand implemented the Cyclic Coordinate Descent method on FPGA, but the Cyclic Coordinate Descent methods are just used in the manipulators with one end-effector [15]. Harish proposed an approach to process the pseudoinverse method on GPU, but the computation of SVD is still processed on CPU at each iteration [16]. As above mentioned, the computation of SVD is incredibly time-consuming and inefficient to parallelize. While the proposed method in our work adopts the transpose method, which is simple and easy to implement on hardware. Based on the simple method, our method can provide outstanding energy efficiency than other methods.

3. INVERSE JACOBIAN METHODS

This section describes the preliminaries of the Inverse Jacobian methods.

For a manipulator, the angles of the n joints can be denoted by an n-dimensional vector $\boldsymbol{\theta}$, and the position of the manipulator's end-effector can be denoted by a 3-dimensional vector $\boldsymbol{X}$. Given the joint angles $\boldsymbol{\theta}$, how to calculate the position $\boldsymbol{X}$ of the end effector is the forward kinematics problem, which is generally represented as:

$$\boldsymbol{X} = f(\boldsymbol{\theta}). \tag{1}$$

On the contrary, the inverse kinematics problem is to find the joint angles $\boldsymbol{\theta}$ for the position $\boldsymbol{X}$, which is denoted by:

$$\boldsymbol{\theta} = f^{-1}(\boldsymbol{X}). \tag{2}$$

In general, the function f is simple, but the function f^{-1} is hard to resolve and the IK problem is usually non-unique or unsolvable [11]. The inverse Jacobian method is a common method for IK problem. In the inverse Jacobian method, the function f is linearly approximated by a Jacobian Matrix $\boldsymbol{J}$. The Jacobian Matrix is denoted by:

$$\boldsymbol{J}(\boldsymbol{\theta}) = \frac{\partial \boldsymbol{X}}{\partial \boldsymbol{\theta}}. \tag{3}$$

Now the change of the position X can be estimated as:

$$\Delta \boldsymbol{X} \approx \boldsymbol{J}\, \Delta \boldsymbol{\theta}. \tag{4}$$

And the change of the joint angles can be estimated as:

$$\Delta \boldsymbol{\theta} \approx \boldsymbol{J}^{-1} \Delta \boldsymbol{X}. \tag{5}$$

So given a target position $\boldsymbol{X}_t$, we can get the difference $\boldsymbol{e}$ between the $\boldsymbol{X} = f(\boldsymbol{\theta})$ and $\boldsymbol{X}_t$, that is $\boldsymbol{e} = \boldsymbol{X}_t - \boldsymbol{X}$. Afterwards, we can obtain the update value $\Delta\boldsymbol{\theta}$, and update the joint angles by:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta \boldsymbol{\theta}. \tag{6}$$

After many iterations, we can get the final joint angles $\boldsymbol{\theta}_t$, which satisfies $\boldsymbol{X}_t \approx f(\boldsymbol{\theta}_t)$.

The above process doesn't elaborate on how to calculate the Jacobian Matrix $\boldsymbol{J}$ and its inverse $\boldsymbol{J}^{-1}$. Now let's briefly describe this question. The Jacobian matrix can be calculated easily by referring to the work [11]. But the matrix $\boldsymbol{J}^{-1}$ is very hard to calculate, because the matrix $\boldsymbol{J}$ may not be invertible or square. So researchers proposed the pseudoinverse method [5, 19, 20] and the transpose method [6, 7]. The pseudoinverse method adopts an iterative approach, singular value decomposition (SVD), to get the approximate pseudoinverse of $\boldsymbol{J}$. In the pseudoinverse method, each iteration needs to calculate the SVD in serial, which will consume a long time. In contrast, the transpose method uses the transpose of $\boldsymbol{J}$ instead of $\boldsymbol{J}^{-1}$, avoiding to compute SVD. In the transpose method, the Equation (5) can be rewritten as:

$$\Delta \boldsymbol{\theta} \approx \alpha\, \boldsymbol{J}^{T} \Delta \boldsymbol{X}. \tag{7}$$

The parameter α is an appropriate scalar, which has a direct impact on the convergence. In the next section, we will discuss how to choose the value of α for fast convergence.

4. QUICK-IK ALGORITHM

Compared with the pseudoinverse method, the transpose method takes too many iterations to converge, so we firstly accelerate the transpose method through reducing the number of iterations.

In the previous section, there still remains how to decide the parameter α in the Equation (7). Samuel [11] proposed a reasonable way:

$$\alpha = \frac{\boldsymbol{e} \cdot \boldsymbol{J}\boldsymbol{J}^{T}\boldsymbol{e}}{\boldsymbol{J}\boldsymbol{J}^{T}\boldsymbol{e} \cdot \boldsymbol{J}\boldsymbol{J}^{T}\boldsymbol{e}} \tag{8}$$

However, the proposed method just gives a near-optimal value for α, which leads limited acceleration for the convergence speed. In this paper, we will search a better α to boost the convergence via a parallel search approach, as shown in Algorithm 1, Line 4 ~ Line 13. At first, a base value α_{base} is given by Equation (8). Then

generate multiple speculative values ($\alpha_1, \alpha_2, ..., \alpha_n$) around the base value α_{base}. For each speculative α_k, we compute the corresponding $\Delta\boldsymbol{\theta}_k$, $\boldsymbol{\theta}_k$ and the position $f(\boldsymbol{\theta}_k)$. After speculative searches in parallel, we can get the joint angles $\boldsymbol{\theta}_o$, where $f(\boldsymbol{\theta}_o)$ is closest to the target position $\boldsymbol{X_t}$ among all the speculative values. Through this method, the IK can select a more optimal α to converge to $\boldsymbol{X_t}$ at each iteration. As shown in Algorithm 1, the parallel search approach will be very expensive if serialized. But at each iteration in the parallel search approach, every speculative process is independent, so these speculative processes can be executed in parallel. It's efficient to run these speculative processes in the multithreaded architectures, e.g. multicore CPU, GPU or special hardware.

Speculation strategy. In Quick-IK algorithm, an important issue is how to generate the speculative values around the base value α_{base}. The work [11] indicated that, for a sufficiently small $\alpha > 0$, the magnitude of the error vector $\boldsymbol{e}$ (among $\boldsymbol{e} = \boldsymbol{X_t} - \boldsymbol{X}$) will decrease. So the speculative value should be set as small as possible. However, if an extremely small value is selected for α, the update value of joint angles $\Delta\boldsymbol{\theta}$ will also be too small, which will show down the convergence seriously. So we set the speculative space as (0, α_{base}], and calculate the k^{th} speculative value as follows:

$$\alpha_k = \frac{k}{Max}\alpha_{base}. \tag{9}$$

In Equation (9), the *Max* is the maximum number of speculations. So there is no speculative value larger than α_{base} and the minimum speculative value is just 1/Max of α_{base}. Through the speculation strategy, the number of speculations will be reduced effectively.

ALGORITHM 1: Quick-IK Algorithm

Input: $\boldsymbol{X_t}$: Target Position
Output: $\boldsymbol{\theta}$: Updated Joint Angles
1. Set $\boldsymbol{\theta}$ through *Random*;
2. **for** *iter* from 1 to *MaxIterations* **do**
3. Compute Jacobian Matrix $\boldsymbol{J}$;
4. $\Delta\boldsymbol{\theta}_{base} = \boldsymbol{J}^T(\boldsymbol{X_t} - f(\boldsymbol{\theta}))$;
5. Compute the base value α_{base} through Equation (8);
6. **for** *k* from 1 to *Max* **do**
7. Speculate: $\alpha_k = (k/Max)\alpha_{base}$;
8. $\Delta\boldsymbol{\theta}_k = \alpha_k \Delta\boldsymbol{\theta}_{base}$;
9. $\boldsymbol{\theta}_k = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}_k$;
10. $\boldsymbol{X_k} = f(\boldsymbol{\theta}_k)$;
11. Error: $error_k = \| \boldsymbol{X_t} - \boldsymbol{X_k} \|$;
12. **if** $error_k < threshold$ **then**
13. return $\boldsymbol{\theta}_k$;
14. **end**
15. **end**
16. Find the $\boldsymbol{\theta}_o$ with the minimum $error_o$;
17. Update Joint Angles: $\boldsymbol{\theta} = \boldsymbol{\theta}_o$;
18. **end**
19. return $\boldsymbol{\theta}$;

5. THE ARCHITECTURE OF IKACC

In this section, we will describe a novel hardware design to implement Quick-IK algorithm. As mentioned above, the multithreads architecture can efficiently parallelize the multiple speculative searches, but the existing multithreaded architecture cannot explore the parallelism in each speculative search and deal with the other serial operations efficiently.

5.1 Overview

Figure 2 describes the architecture of IKAcc, which consists of four modules: Serial Process Unit (SPU), Speculative Search Units (SSU), Parallel Search Scheduler and Speculation Selector.

Serial Process Unit (SPU): As shown in Algorithm 1, Line 3 ~ Line 5, the processor must calculate the Jacobian Matrix $\boldsymbol{J}$, the base update-value $\Delta\boldsymbol{\theta}_{base}$ of the joint angles and α_{base} in the beginning of each iteration. These processes have data dependence, it's hard to process in multiple independent threads. In IKAcc, the serial process is decomposed and recomposed for pipelined execution, detailed in next section. Through this method, IKAcc speeds up the serial process, but also avoids the store of intermediate results.

Speculative Search Units (SSU): In Quick-IK, the speculative search for each speculative value is independent at one iteration. In IKAcc, each SSU processes a speculative parameter value α_k independently and all the units can process the speculative searches at same time. At first, the SSU generates a speculative value α_k based on the α_{base} and gets the new joint angles $\boldsymbol{\theta}_k$ by Algorithm 1, Line 8. Then calculate the forward kinematics of $\boldsymbol{\theta}_k$, which is the main process for each speculative search. SSU adopts a Forward Kinematics Unit (FKU) to process the forward kinematics at fine-grained parallelism.

Parallel Search Scheduler: As above mentioned, there are a bunch of Speculative Search Units to process at the same time, so the Parallel Search Scheduler broadcasts the results of SPU, $\boldsymbol{\theta}$, $\Delta\boldsymbol{\theta}_{base}$, and α_{base}, to all the Speculative Search Units. In addition, the number of Speculative Search Units may not match the number of speculations in algorithm due to the limited resources in hardware. When the speculations in algorithm is more than the number of SSUs, each SSU will process multiple speculative searches. So the Parallel Search Scheduler also manages the mapping of speculations to SSUs. The Parallel Search Scheduler schedules *MaxSSUs* speculations to SSUs at one time, among the *MaxSSUs* is the number of SSUs in IKAcc. In one schedule, a SSU just processes one speculative search. After the SSUs complete the speculations in each schedule, Parallel Search Scheduler will schedule the next *MaxSSUs* speculations to SSUs. After multiple schedules, all the speculative searches will be processed by the limited hardware.

Parameter Selector: The unit just selects the $\boldsymbol{\theta}_o$ with minimum error $error_o$ from multiple speculations, shown in Algorithm 1, Line 16. Due to the mismatch between the speculations in software and hardware, the Parameter Selector needs to store and compare the last result at each schedule, but the overhead is negligible.

5.2 Explore Parallelism of Each Speculation

The speculative searching can reduce the number of iterations via processing in parallel efficiently. And there remains space to accelerate Quick-IK. At each speculative search, the main process is the calculation of forward kinematics, which mainly operates 4x4 matrix multiplication, as follows.

$$f(\boldsymbol{\theta}) = \prod_1^N {}^{\boldsymbol{i-1}}\boldsymbol{T_i} \tag{10}$$

In Equation (10), ${}^{\boldsymbol{i-1}}\boldsymbol{T_i}$ is a 4x4 matrix of the i^{th} joint angle, called transformation matrix.

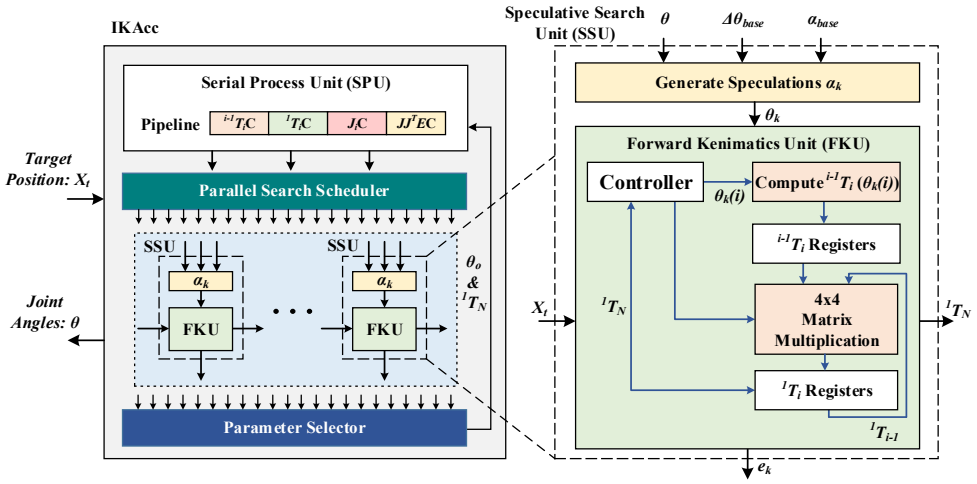


Figure 2. The architecture of IKAcc.

In the matrix multiplication, each entry of the result can be calculated independently, so it's efficient to process the matrix multiplication in parallel. As we all have known, GPU can process the large matrix multiplication perfectly. But in forward kinematics for any robots or manipulators, the sizes of the transformation matrixes are fixed and small, which is not efficient to process on GPU, where the number of the threads with same instruction is not less than 32. While the 4x4 matrix multiplication just needs 16 threads in parallel. In IKAcc, we adopt a logic block to process 4x4 matrix multiplication in parallel, as shown in Figure 2. To implement the matrix multiplication, we can adopt a logic block with many multipliers and adders to calculate each entry of the result matrix, but this implementation will introduce too much overhead. By means of high-level synthesis tool, we get an efficient logic block, which adopt a few of multipliers and adders to calculate the result in tens of cycles.

5.3 The Optimal Process for Serial Process

In the beginning of each iteration, it needs to compute the Jacobian Matrix $\boldsymbol{J}$ and the base value α_{base}. These steps will take a long time to work out the results and consume a large amount of memory resources to store the intermediate results, shown in Figure 3 (a). For example, to calculate the Jacobian Matrix $\boldsymbol{J}$, it's necessary to calculate and store the set of $^1\boldsymbol{T_i}$. After the computation of $\boldsymbol{J}$, the algorithm will store the $3\times N$ Matrix $\boldsymbol{J}$, used to calculate $\boldsymbol{JJ^TE}$. So it's necessary to optimize these processes.

To optimize the process for Quick-IK, we firstly combine multiple loops with data dependence, as shown in Figure 3 (a) and (b). For the i^{th} iteration, we can sequentially calculate $^{i-1}\boldsymbol{T_i}$, $^1\boldsymbol{T_i}$ and $\boldsymbol{J_i}$. The combination is obvious. When calculate $\boldsymbol{J_i}$, the $^1\boldsymbol{T_N.P}$ is from the speculative search at the last iteration, shown in Algorithm 1, Line 9.

For the computation of $\boldsymbol{JJ^TE}$, we adopt the method as follows:

$$\boldsymbol{JJ^TE} = \sum_{i=1}^{N} \boldsymbol{J_i J_i^T E} \qquad (11)$$

Through this method, the computation of $\boldsymbol{JJ^TE}$ can also be combined with $^1\boldsymbol{T_i}$ and $\boldsymbol{J_i}$ in one iteration.

In the combination loop, one iteration can be processed in pipeline, as shown in Figure 3 (c). Each iteration is divided into four stage pipeline: $^{i-1}\boldsymbol{T_iC}$, $^1\boldsymbol{T_iC}$, $\boldsymbol{J_iC}$ and $\boldsymbol{JJ^TEC}$. Each stage can forward its results to next stage directly, so it does not need to store the intermediate results to memory or registers.

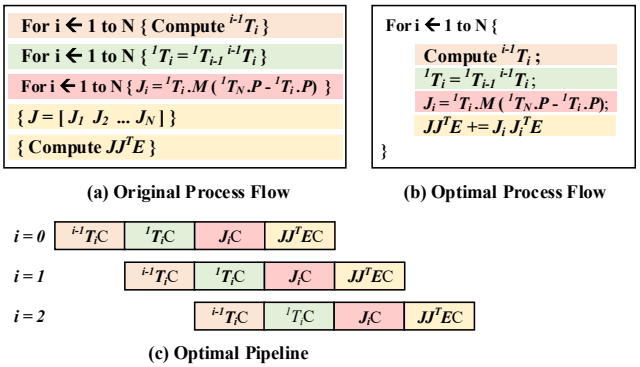


Figure 3. The optimal process for Serial Process.

6. EVALUATION

In this section, we evaluate the efficiency of the proposed approach Quick-IK and the novel architecture IKAcc on multiple manipulators with various degrees of freedom. At first, we evaluate the efficiency on reducing the number of iterations through the speculative search approach in parallel. Second, we evaluate the performance and energy efficiency of Quick-IK on different architectures, Atom CPU, NVIDIA Tegra X1 (TX1) GPU and IKAcc. For robotic system, the power is another critical constraint, especially the mobile robot, such as LittleDog equipped with lithium polymer batteries [14]. Atom and TX1 can provide higher energy efficiency than server CPUs and GPUs. They are perfect to use in robots or manipulators.

6.1 Experimental Setup

To evaluate the effect of iteration reduction, we implement the original transpose method (JT-Serial) and Quick-IK using C++ on Intel Atom D2500 1.86GHz, and the SVD-based pseudoinverse method (J-1-SVD), which is the optimal method in serial process [20]. The implementation of the pseudoinverse method is from the Kinematics and Dynamics Library (KDL), which has been integrated in ROS [13].

The evaluation of performance and energy efficiency is implemented via three different methods: the original transpose method, Quick-IK and the based-SVD pseudoinverse method. In

addition, the Quick-IK is also implemented on three different architecture (i) Mobile CPU implementation: Intel Atom, C++, (ii) Mobile GPU implementation: NVIDIA Jetson X1, CUDA C, (iii) IKAcc implementation. For the implementation of IKAcc, we adopt the high-level synthesis tool VivadoHLS to generate the RTL of IKAcc. We further adopt Synopsis Design Compiler to evaluate the performance and area of IKAcc with the Nangate 65nm Open Cell Library. In the evaluation of energy efficiency, we adopt Synopsis Primetime-PX to perform power analysis.

Accuracy constraint. For IK solvers, it's impossible to obtain the joint angles $\boldsymbol{\theta}_o$ which satisfies $\boldsymbol{X}_t=f(\boldsymbol{\theta}_o)$. In general, IK solvers try to obtain the joint angles $\boldsymbol{\theta}_o$ which satisfies $\|\boldsymbol{X}_t - f(\boldsymbol{\theta}_o)\| < accuracy$. In the evaluation, the *accuracy* is 1E-2 meter. So the above methods won't stop iterating until meet constraint in accuracy or reach the maximum number of iterations (10k in our evaluation).

Table 1. The methods in evaluations.

Platform	Intel Atom	Nvida TX1	IKAcc
Original transpose method	JT-Serial	-	-
Pseudoinverse method	J-1-SVD	-	-
Quick-IK	JT-Speculation	JT-TX1	JT-IKAcc

6.2 Evaluation of Quick-IK

The efficiency of original transpose method is in controversy because of the large number of iterations involved. In this paper, Quick-IK is proposed to mitigate this problem. To evaluate the efficiency of Quick-IK, we implement the original transpose method, Quick-IK and the pseudoinverse method on multiple manipulators with various degrees of freedom, 12-DOF, 25-DOF, 50-DOF, 75-DOF and 100-DOF.

For Quick-IK, the number of speculations has an impact on the convergence. As shown in Figure 4, the number of iterations declines with the increase of speculations. But the larger number of speculations will consume more resources, e.g. hardware, energy or time. We will trade-off between the speculations and resources. The results show that 64 speculations may be a great choice. Compared with 64 speculations, 128 speculations cannot significantly speed up the convergence under various DOF manipulators. For the following evaluation, we will set the number of speculations as 64.

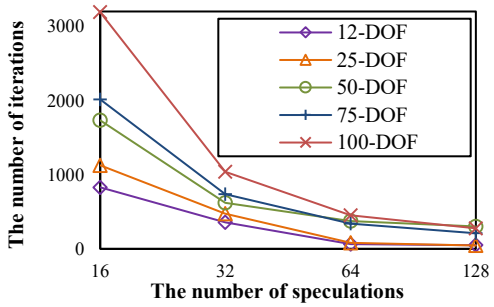
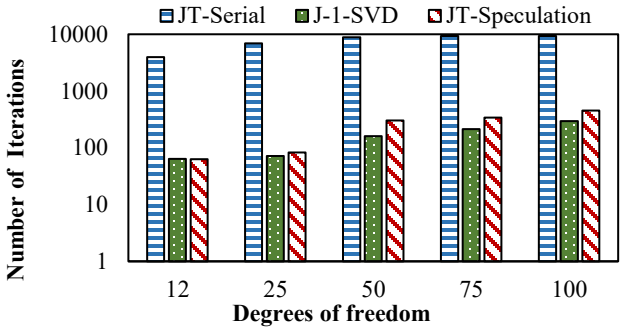
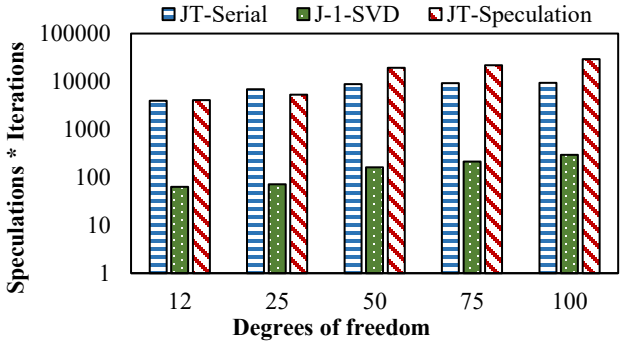


Figure 4. The number of iterations under different number of speculations.

Figure 5 (a) shows the number of iterations taken by different IK methods under various DOF manipulators. For each DOF configuration, every method will try to solve 1K target positions. As shown in result, the proposed approach Quick-IK in this work can reduce the number of iterations to the comparable level of the pseudoinverse method. Compared with the original transpose method, Quick-IK can reduce the number of iterations by 97%. Figure 5 (b) shows that the computation load using the different methods. The result shows that Quick-IK has similar computation load with the original transpose method, so Quick-IK cannot reduce the computation than original transpose method, even introduce more load. But the speculations can be parallelized on the multithreads architecture, so the computing time can be decreased via more hardware resources. Through speculative searching in parallel, the transpose method can be used as a real-time IK solver.



(a) Number of iterations under various DOF manipulators.



(b) Computation load under various DOF manipulators. For JT-serial and J-1-SVD, the speculation is one.

Figure 5. The iterations under various DOF manipulators.

6.3 Evaluation of IKAcc

In this section, we will evaluate the proposed architecture IKAcc in performance, power and Energy-efficiency. In the evaluation, the number of speculations is also 64 in software, but IKAcc contains only 32 Speculative Search Units (SSU), so it needs two schedules by the Parallel Search Scheduler. For the implementation of GPU (JT-TX1), the speculative searches are processed on GPU of TX1 and the serial process is executed on the A57 CPU of TX1.

6.3.1 Performance

The results, shown in Table 2, indicate that the GPU implementation of Quick-IK is faster than the original transpose method JT-Serial about 40x. But compared with the pseudoinverse method, the improvement is limited, just about 3x. The main reason is that GPU needs to exchange data with CPU at each iteration. But if process the entire algorithm in GPU, the time will be longer due to the low ability of GPU in serial execution. So it's necessary to design special logic to process the serial block. As the results shown, IKAcc can accelerate the transpose method by 1700x. Compared with the GPU

implementation, IKAcc is also 30x faster. IKAcc can solve IK within 12 milliseconds for 100-DOF manipulator. While the pseudoinverse method, will take 1.38 seconds to solve. So IKAcc can provide more real-time for robots and manipulators.

Table 2. The performance under various IK methods and architectures. The average time (ms) under 1K solutions.

Platform	Atom			Nvidia TX1	IKAcc
DOF	JT-Serial	J-1-SVD	JT-Speculation	JT-TX1	JT-IKAcc
12	622.05	96.76	288.06	38.30	0.3042
25	2330.53	144.57	656.15	47.91	0.8243
50	6010.24	469.87	5285.14	185.18	4.5373
75	9570.49	637.57	7704.93	217.91	7.6572
100	12990.81	1382.35	12383.25	311.74	12.1125

6.3.2 Power and Energy-efficiency

In Table 3, we can see that the average power of IKAcc is just 158.6mW @1V,1GHz, while the mobile CPU Atom's power is about 10W and the TX1's average power is 4.8W. So IKAcc can be used in more robots or manipulators with power constraints.

The energy consumption of the original transpose method isn't acceptable, which is over tens of J. The pseudoinverse method, processed on Atom, consumes about 1J energy for 12-DOF manipulator, even worse for 100-DOF manipulator. The TX1 may consume 1.49J for 100-DOF manipulator. While IKAcc just consumes about 1.92mJ for a manipulator with 100 DOFs. The improvement of energy efficiency is outstanding. The proposed architecture can achieve energy-efficiency 776x that of TX1 and 5200x that of Atom using the pseudoinverse method. Compared with the original transpose method, the improvement will be more tremendous.

Table 3. The details of various hardware platforms.

Platform	Intel Atom	Nvidia TX1	IKAcc
Technology	32nm	20nm	65nm 1.1V
Frequency	1.86GHz	up to 1.9GHz	1GHz
Average Power	10W	4.8W	158.6mW
Area	-	-	2.27mm^2

7. CONCLUSIONS AND FUTURE WORK

In this paper, we present a nearly brute-force search approach to reduce the number of iterations for the transpose IK method. The proposed method Quick-IK takes full advantage of hardware to improve performance. Compared with the original transpose IK method, Quick-IK can reduce the number of iterations by 97%, which enables the transpose method as a real-time solver for inverse kinematics. To further improve the performance and energy efficiency of Quick-IK, we propose a novel architecture IKAcc, which can achieve 30x performance and 776x energy efficiency than TX1 GPU. With the outstanding performance and energy efficiency, IKAcc will be a promising solver for inverse kinematics.

8. ACKNOWLEDGMENTS

This work is supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 61522406, 61532017, 61521092 and 61504153. The corresponding authors are Yinhe Han and Ying Wang.

9. REFERENCES

[1] BigDog. [Online]. Available: http://www.bostondynamics.com/robot_bigdog.html, accessed 2016.

[2] The Perfect Match. [Online]. Available: https://www.kuka.com/en-de/about-kuka/brand/timo-boll-the-perfect-match, accessed 2016.

[3] Valkyrie. [Online]. Available: http://www.nasa.gov/feature/valkyrie, accessed 2015.

[4] L. Wang, and C. Chen, "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," In *IEEE Transactions on Robotics and Automation*, vol. 7(4), pp. 489-499, 1991.

[5] D. E. Whitney, "Resolved Motion Rate Control of Manipulators and Human Prostheses," In *IEEE Transactions on Man-Machine Systems*, vol. 10(2), pp. 47-53, 1969.

[6] W. A. Wolovich and H. Elliott, "A computational technique for inverse kinematics," In *IEEE Conference on Decision and Control*, pp. 1359-1363, 1984.

[7] J. E. Slotine, "The robust control of robot manipulators," In *The International Journal of Robotics Research*, vol. 4(2), pp. 49-64, 1985.

[8] A. D'Souza, S. Vijayakumar and S. Schaal, "Learning inverse kinematics," In *IEEE/RSJ Conference on Intelligent Robots and Systems*, vol. 1, pp. 298-303, 2001.

[9] H.P. Singh and N. Sukavanam, "Neural network based control scheme for redundant robot manipulators subject to multiple self-motion criteria," In *Mathematical and Computer Modelling*, vol. 55(3-4), pp. 1275-1300, 2012.

[10] Y. Xia and J. Wang, "A dual neural network for kinematic control of redundant robot manipulators," In *IEEE Transactions on System s, Man, and Cybernetics*, vol. 31(1), pp. 147-154, 2001.

[11] S. R.Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," In *IEEE Journal of Robotics and Automation*, vol. 17, pp. 1-19, 2004.

[12] D. Kalman, "A singularly valuable decomposition: the SVD of a matrix," In *the college mathematics journal*, vol. 27(1), pp. 2-23, 1996.

[13] Kdl. [Online]. Available: http://wiki.ros.org/kdl, accessed 2016.

[14] LittleDog. [Online]. Available: http://www.bostondynamics.com/robot_littledog.html, accessed 2016.

[15] D. Hildenbrand, H. Lange, F. Stock and A. Koch, "Efficient inverse kinematics algorithm based on conformal geometric algebra using reconfigurable hardware", 2008.

[16] P. Harish, M. Mahmudi, B. L. Callennec, and R. Boulic, "Parallel Inverse Kinematics for Multithreaded Architectures," In *ACM Transactions on Graphics*, vol. 35(2), 2016.

[17] A. Goldenberg, B. Benhabib and R. Fenton, "A complete generalized solution to the inverse kinematics of robots," In *IEEE Journal on Robotics and Automation*, vol. 1(1), pp. 14-20, 1985.

[18] S. Farzan and G. N. DeSouza, "From D-H to inverse kinematics: A fast numerical solution for general robotic manipulators using parallel processing," In *IEEE/RSJ Conference on Intelligent Robots and Systems*, pp. 2507–2513, 2013.

[19] A. Olaru, S. Olaru and N. Mihai, "Application of a new Iterative pseudo-inverse Jacobian Neural Network Matrix technique for controlling geckodrive DC motors of manipulators," In *RSI Conference on Robotics and Mechatronics*, pp. 114-120, 2015.

[20] S. R. Buss and J. S. Kim, "Selectively damped least squares for inverse kinematics," In *journal of graphics, gpu, and game tools*, vol. 10(3), pp. 37-49, 2005.

[21] Y. S. Kung, B. T. H. Linh, M. K. Wu, F. C. Lee and W. C. Chen, "FPGA-Realization of Inverse Kinematics Control IP for Articulated and SCARA Robot," In *Springer Design and Computation of Modern Engineering Materials*, pp. 205-213, 2014.