Available online at www.sciencedirect.com
SCIENCE DIRECT®

Future Generation Computer Systems 22 (2006) 80–87

FGCS
FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Bioinformatic searches using a single-chip shared-memory multiprocessor[☆]

Scott F. Smith[a,*], James F. Frenzel[b]

[a] Department of Electrical and Computer Engineering, Boise State University, Boise, ID 83725, USA
[b] Department of Electrical and Computer Engineering, University of Idaho, Moscow, ID 83844, USA

Received 5 November 2003; accepted 5 November 2003
Available online 7 December 2004

Abstract

A single-chip shared-memory multiprocessor architecture is introduced which is particularly well suited to common bioinformatic computing tasks. The architecture uses asynchronous bus interfaces to create an integrated circuit design methodology allowing for scaling of the multiprocessor with very little design effort. A key aspect of this design methodology is that it is not necessary to expend significant design resources and chip area on the clock tree. An analysis of the Smith–Waterman alignment algorithm running on this architecture shows that the performance penalty due to increased bus latency compared to a fully synchronous architecture is negligible.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Bioinformatics; Multiprocessor; Scalable; Asynchronous interface

1. Introduction

It is often desirable to search a protein or genome database for locations that are similar to that of some query sequence. These databases already contain billions of characters of sequence data and the amount of available data are increasing exponentially. Even a simple-minded search that looks for an exact match between the query string and a sub-string of the database is a very computationally demanding task. Unfortunately, such a simple-minded approach is not very useful. Good searches need to allow for the possibility that individual database characters may be mutated or that subsequences have been inserted or deleted from the database. Heuristic approaches such as BLAST and FASTA [1] allow for mutations, insertions, and deletions, but they do not have well defined statistical properties like algorithms based on dynamic pro-

[☆] An earlier version of this work has appeared in the Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications.
[*] Corresponding author. Tel.: +1 208 426 5743; fax: +1 208 426 2470.
E-mail addresses: sfsmith@boisestate.edu (S.F. Smith), jfrenzel@uidaho.edu (J.F. Frenzel).
URL: http://coen.boisestate.edu/ssmith/ (S.F. Smith).

gramming such as Smith–Waterman [2]. Even though Smith–Waterman is technically superior, the required computation time is much longer and hence it is not often used.

There are four common types of processing systems used for sequence alignment processing: general purpose desktop computers, general purpose supercomputers, application-specific integrated circuit (ASIC)-based accelerators, and field-programmable gate array (FPGA)-based accelerators. General purpose desktop computers are typically limited to using algorithms like BLAST or FASTA since Smith–Waterman is too slow. The other three types of processing systems can handle Smith–Waterman.

General purpose supercomputers are often very expensive, but it is the purpose of this paper to look at a general purpose supercomputer that is likely to be much less expensive. ASIC-based accelerators have the disadvantage that the production volume for the ASIC will be small and therefore the design costs cannot be spread over very many units. Examples of this type of accelerator are Kestrel [3], SAMBA [4], and GeneMatcher 2 [5]. The performance of Kestrel is discussed in Section 3, SAMBA is a rather old design (1997), and GeneMatcher 2 is proprietary, so not much information exists on it's detailed design. FPGA-based accelerators are much less expensive to design, but the unit cost of the FPGAs (which are bought as a standard part from an FPGA vendor) is rather high. An example of an FPGA-based accelerator is DeCypher [6], which is a proprietary design.

This paper looks at the design of a multiprocessor system that has the amount of computing power necessary to run the Smith–Waterman algorithm on a single integrated circuit in a reasonable amount of time. This system is a general purpose shared-memory multiprocessor, which is usable in many applications. The goal here is to show that the multiprocessor is a very efficient design for this important bioinformatics problem, but is not intended to imply that the architecture is in any way specially designed for the problem. A key feature of this single-chip multiprocessor is that it scales well as integrated circuit process technology improvements allow for more processors on a chip.

The idea is to put many small and simple processors on a large integrated circuit. A system bus allows data to pass between the processors and a shared memory. For the example application, this shared memory system may actually only be a cache for a much larger memory located off the integrated circuit. The database items are only read once and each result is only written once, so traffic between off-chip memory and on-chip shared memory cache will be low. For other applications, a larger on-chip shared memory may be desirable.

The processors, segments of the system bus, and memory systems are all intended to be existing intellectual property (IP) designs. A custom interface between bus segments has been designed, but does not need redesign for each multiprocessor implementation. The processors used in this example application are ARM9 processors with 8 KB of instruction cache and 8 KB of data cache (an ARM922T CPU core). The segments of the system bus are composed of standard Advanced Microcontroller Bus Architecture, Advanced High-performance Bus (AMBA AHB) [7]. These buses are multi-master and use no bi-directional signals. The design methodology is not limited to ARM processors or the AMBA bus, they are only used as examples.

The major advantage of the design method shown in this paper is that it can be implemented with very little design effort. The hardest part of composing large integrated circuits using pre-existing IP is to obtain timing closure at high clock frequencies. Creating a low-skew clock tree is very time consuming and the result requires large amounts of chip area and power. One alternative is to use local clocking with asynchronous interfaces such that low-skew clocks only need to be designed for much smaller areas. Systems with this clocking scheme have been called *globally-asynchronous locally-synchronous* (GALS) as originally proposed in [8].

Much of the GALS literature (for example, [9] and [10]) focuses on asynchronous "wrappers" surrounding synchronous logic, where the asynchronous wrappers can stop the local clock. This is unappealing as a generic technique since it interacts badly with some types of synchronous logic design (e.g., dynamic logic). It also will require analysis to assure that deadlock cannot occur. The asynchronous interfaces used in this paper do not require alteration of any local clock. The interfaces have been shown to work with latency less than 1.1 ns in a 1.8 V, 180 nm process using SPICE simulation. Details of the interface operation and SPICE simulation results can be found in [11].