Journal of Discrete Algorithms 28 (2014) 23–30

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Layouts for improved hierarchical parallel computations

Michael Hirsch [a], Shmuel T. Klein [b,*], Yair Toaff [a]

[a] IBM – Diligent, Tel Aviv, Israel
[b] Computer Science Department, Bar Ilan University, Ramat Gan 52900, Israel

ARTICLE INFO

Article history:
Available online 3 July 2014

Keywords:
Data compression
Parallel processors
Modular arithmetic

ABSTRACT

New layouts for the assignment of a set of n parallel processors to perform certain tasks in several hierarchically connected layers are suggested, leading, after some initialization phase, to the full exploitation of all of the processing power all of the time. This framework is useful for a variety of string theoretic problems, ranging from modular arithmetic used, among others, in Karp–Rabin type rolling hashes, as well as in cryptographic applications, and up to data compression and error-correcting codes.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

We consider an (unbounded) stream of character strings of fixed, given length k, called *chunks*, and wish to apply a certain operation on each of the elements of this stream. For the ease of description, we shall use the remainder operation modulo some large integer P as a running example, considering each chunk as representing an integer of size k bytes. The method, however, applies as well to a large variety of other associative operations: if one considers a chunk as a sequence of numbers, one could calculate their sum, product, maximum or minimum, or Boolean operations like AND, OR or XOR.

The motivation for the repeated application of the remainder operation stems from our work on a large deduplication system [1], whose technical details are not relevant here. For the present work, it suffices to know that we wish to evaluate the remainders, modulo a large prime number P, of an unbounded sequence of input chunks. Specifically, we shall: (1) identify a chunk B, which is a character string of fixed size k, with its ASCII encoding; (2) consider this encoding as the standard binary representation of a large $8k$-bit long integer; and (3) evaluate $h(B) = B \bmod P$.

The use of the remainder operation has many other applications, beside deduplication, like Karp and Rabin's probabilistic pattern matching algorithm [8], modular exponentiation in cryptographic methods, like El Gamal's scheme [4] or RSA [13].

The length k of the chunks may be 512 or more, so that the evaluation might put a serious burden on the processing time. This can be improved if we assume the availability of several processors working in parallel. We show below how to exploit a set of n hierarchically connected parallel processors to perform the needed operations in several layers, but keeping all the processors busy without idle time, after some initialization phase. The challenge is to design the transition protocols from one step to another in a way that can be repeated indefinitely.

Parallelism has been discussed in connection with accelerating hashes in deduplication systems in [11], which uses the cryptographic SHA hash function for collision resistant fingerprinting, and in [14], presenting a deduplication system called P-Dedupe that pipelines and parallelizes the processes. More generally, Gharaibeh et al. [5] study the use of hashing in storage systems when a Graphics Processing Unit (GPU) can be used to speed up the processing. In another application, Collange et al. [3] use GPUs to parallelize hashing for the detection of image fragments in an image database.

* Corresponding author.
E-mail addresses: mikizvi@gmail.com (M. Hirsch), tomi@cs.biu.ac.il (S.T. Klein), yair.toaff@gmail.com (Y. Toaff).

http://dx.doi.org/10.1016/j.jda.2014.06.002
1570-8667/© 2014 Elsevier B.V. All rights reserved.

Other applications of the hierarchical method we describe below, beside remainder calculations, are the compression of sparse bit-strings, as described in [2] in which the recursive operation is the bit-wise OR, and the evaluation of the parity bits in the Hamming Error-Correcting Code [6], where the recursive operation is summation modulo 2, or equivalently, bit-wise XOR. The general case of a parallel hierarchical evaluation of an associative operation has been studied in [10], and our basic scheme with $\log n$ phases for n elements is mentioned in [9], but without referring to the layout permitting to keep all processors busy. Parallel implementations for more specific operations (multiplication and addition modulo $(2^n \pm 1)$) appear in [15].

The pertinence of the current work to string manipulation methods is thus twofold: it is not restricted to the suggested solution itself, which assigns the processors on the basis of the binary representation of their indices, but extends also to a large body of potential string-theoretic applications, such as data compression, pattern matching, error-correcting codes, cryptography, and others.

In the next section, we introduce the notation used below followed by the details of the suggested layouts in Section 3, using the application to the evaluation of a modulus B mod P as an underlying example, rather than giving a generic description of the method. Section 4 deals with the mathematical details of this application and Section 5 presents some experimental results.

2. Notation

Given a sequence of chunk $B^i = x_1^i x_2^i \cdots x_m^i$, where the x_j^i denote characters of an alphabet Σ, we wish to apply the classical hash function $h(B^i) = B^i \bmod P$ for some large prime number P. We assume the availability of several processors working in parallel. A simplistic solution of assigning a set of ℓ processors would be as follows. Suppose an (unbounded) stream of non-overlapping chunks B is given, we shall process them by subsets of ℓ elements. For each subset, the ℓ processors are assigned sequentially to the ℓ chunks. Once all of them have produced their output, the whole set of processors is reassigned to the following ℓ chunks, etc. We may assume that all the processors need roughly the same time for the processing of their respective chunks, since the execution time of the given operation, B mod P in our example, is not data dependent. Therefore this way of processing the stream by subsets of ℓ chunks keeps all the processors busy all of the time. We call this the *basic* parallel method.

The drawback, on the other hand, is that results are produced by packets of ℓ every t time units, where t is the sequential time needed by a single processor for processing a single chunk. In a streaming mode, we would prefer to apply the combined power of several processors in parallel on a *single* chunk and thereby obtain the requested result in less than t time units.

The following strategy could thus be applied. Partition the chunk to be processed into n *blocks* of k/n bytes each. For example, a chunk of 512 bytes could be split into $n = 64$ blocks of 8 bytes each. In a first stage which we call Step 0, a set of n processors is used to work simultaneously on the n blocks of the chunk. In Step 1, only $n/2$ processors are used, each acting on two blocks evaluated in the previous step, and in general in Step i, only $n/2^i$ processors are used, each acting on two blocks evaluated in the previous step $i-1$. Finally, in Step $\log n$, only a single processor is used. While the overall work of all the processors together is not reduced relative to an equivalent sequential evaluation on a single processor, the total processing time, if one accounts only once for commands executed in parallel, is reduced from $t = O(n)$ to $t = O(\log n)$. We call this the *hierarchical* method.

However, only in the first stage is the set of processors fully exploited, and in fact, for reasonable choices of n, most of the processors remain idle for most of the time. The average number of occupied processors is

$$\frac{n + \frac{n}{2} + \frac{n}{4} + \cdots + 2 + 1}{1 + \log n} = \frac{2n - 1}{1 + \log n},$$

which means that for $n = 64$, only about 28% of the processors are busy on average. The present work addresses this waste of processing time, by grouping several tasks together so as to get full exploitation of the available processing power, thereby reducing the waste to zero. This optimal utilization of the n processors is achieved by means of a particular strategy and a special way, to be described below, of assigning processors to tasks; we shall refer to a specific processor assignment as a *layout*.

The main idea leading to the full exploitation of all of the processors all of the time, is to assign them in such a way that, after an initialization phase of $\log n$ steps, a sequence of $\log n$ consecutive chunks will be processed simultaneously in parallel. The challenge is therefore to design an appropriate layout, showing how to assign the available processors at each time step. In particular, this layout has to be consistent over time transitions from step i to step $i+1$, while also complying with the hierarchical definition of the function to be evaluated. In other words, we are looking for a function from the set of indices of the processors to itself, showing how to assign the subtasks to the processors at each step, so that the chain of transitions can be continued indefinitely without wasting any processing power. This will be called the *interleaving hierarchical* method.

Download English Version:

https://daneshyari.com/en/article/431278

Download Persian Version:

https://daneshyari.com/article/431278

Daneshyari.com