Clock Synchronization in Cell BE Traces

Marina Biberstein, Yuval Harel, and Andre Heilper

IBM Haifa Research Lab
Haifa University Campus
Haifa 31905, Israel
{biberstein,harely,heilper}@il.ibm.com

Abstract. Cell BE is a heterogeneous multicore processor that has been developed as a means for efficient execution of parallel and vectorizable applications with high computation and memory requirements. The transition to multicores introduces the challenge of providing tools that help programmers tune their code running on these architectures. Tracing tools, in particular, often help locate performance problems related to thread and process communication.

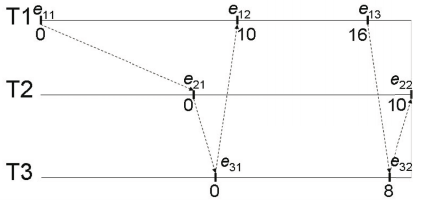
A major impediment to implementing tracing on Cell is the absence of a common clock that can be accessed at low cost from all cores. The OS clock is costly to access from the auxiliary cores and the hardware timers cannot be simultaneously set on all the cores. In this paper, we describe an offline trace analysis that assigns wall-clock time to trace records based on their thread-local time stamps and event order. Our experiments on several Cell SDK workloads show that the indeterminism in assigning the wall-clock time is low, on average 20–40 clock ticks (1.4–2.8 μs for 14.8 MHz clock). We also show how various practical problems, such as the imprecision of time measurement, can be overcome.

1 Introduction

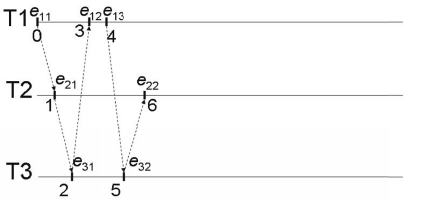
The Cell BE [1] has been developed as a power-efficient processor for running highly parallel and vectorizable workloads. A heterogeneous multicore, it has one 64-bit Power Architecture core, known as the *Power Processor Element* (PPE), and eight specialized single-instruction multiple-data (SIMD) coprocessors called *Synergistic Processor Elements* (SPEs). The SPEs operate from a local store, containing both the code and the data, and communicate with the main memory using the *Direct Memory Access* (DMA). Additional channels of communication between threads executing on different cores include mailboxes and mutual exclusion routines.

Tracing tools are a popular answer to performance and correctness problems encountered by parallel programs. When implementing trace collection and visualization on Cell [2], one of the major obstacles we encountered was the lack of a wallclock, i.e., a clock efficiently accessible from all the cores. The hardware provides a 64-bit *timebase* register on the PPE and 32-bit *decrementer* registers on the SPEs, which are modified all at the same rate; unfortunately, there is no way to set all these registers simultaneously.

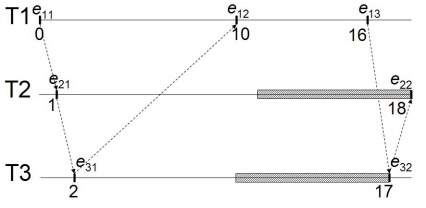
E. Luque, T. Margalef, and D. Benítez (Eds.): Euro-Par 2008, LNCS 5168, pp. 3–12, 2008.



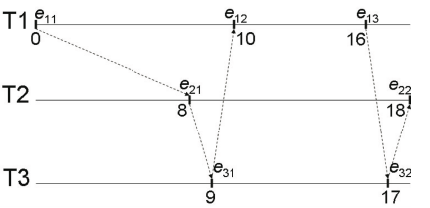
(a) Sample trace



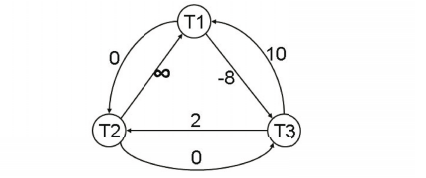
(b) Reconstruction with logical time



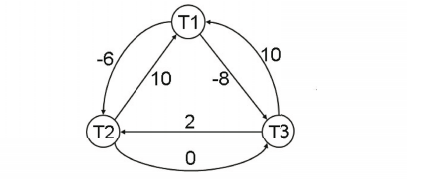
(c) Reconstruction with clock-based logical time



(d) A reconstruction with global time



(e) Constraints graph for (a)



(f) Bounds graph for (a)

Fig. 1. A parallel execution example

Figure 1(a) provides an example of a multi-threaded execution and its trace in the absence of a wall clock. There are three threads in this example, marked T_1, T_2, and T_3. Each of the threads executes several traced events, marked $e_{11}, \ldots, e_{32}$. The trace record of an event identifies the executing thread, the timestamp based on the thread's clock (shown in the figure below each event), and the event's happened-before relationship with events in other threads (shown in the figure with dotted arrows). The information missing from the trace is the relative shift of the threads' clocks, which is required to determine how much time elapsed between events in different threads, e.g., e_{11} and e_{21}. This information is critical for performance analysis and visualization of the trace.

In this paper, we present an algorithm for post-processing the trace and establishing a wall-clock time that is consistent with the original trace, i.e., that preserves both the event precedence and the relative timing of events within each thread. Such wall-clock time for our example is illustrated by Figure 1(d). The algorithm can determine the precision it achieves on the given input. We also discuss technical details of implementing this algorithm for Cell and illustrate its precision on traces collected on several important Cell workloads.

The paper is organized as follows. Section 2 discusses related work. In Section 3 we describe the wall-clock time algorithm. In Section 4 we show how this algorithm can be applied to Cell traces collected by PDT. Section 5 shows the results of algorithm execution on several publicly available benchmarks. Section 6 wraps up with conclusions and future work.

2 Related Work

The problem of clock synchronization is long familiar in the context of debugging parallel programs. Lamport [3] proposed an algorithm for construction of *logical time* based on happened-before relationships between events in different threads. This algorithm is only concerned with reconstructing a possible event order; the thread-local timestamps are not even part of its input. The result of the execution of this algorithm on our sample trace is shown in Figure 1(b), with the corresponding logical time marked below each event. While logical time and its variants such as vector time [4] found a wealth of applications, from distributed debugging to coherency protocols [5], they are clearly inapplicable to the performance analysis.

Lamport [3] also offers a variant of the logical time algorithm that aims to preserve, to some extent, the timing of the traced execution. It bumps the logical timestamps enough to avoid shrinking the interval between the event and its predecessor in the same thread. The objective of this algorithm is to compensate for clock drift and small imprecision in clock synchronization. It does not cope well with clocks that are not synchronized at all. Figure 1(c) shows the algorithm's output on input from Fig. 1(a). Since e_{21} is timed too early, the interval between e_{11} and e_{21} takes one time unit instead of eight, and the interval between e_{21} and e_{22} is correspondingly longer; thus even interval durations within the thread are not preserved.

Another common means for clock synchronization is the Network Time Protocol (NTP) [6]. This protocol solves both clock offset and clock drift problems, the latter of which is irrelevant in the Cell context. There were several reasons that made the adoption of an NTP-style solution impractical. The SPEs would have to monitor some communication channel for the arrival of time data from the PPE, introducing the tradeoff between low precision and high overhead. Additionally, because of the SPE context switches, which affect the clock offsets, each client clock may run only for a relatively short time between resets, and will not be aware of its own resets, again forcing the synchronizer to increase overhead to achieve reasonable precision. Other approaches, like [7], use the constraints to set up an over-determined set of equations, then compute a least-squares optimal solution, which may not prevent incosistent time sequences.

The all-pairs shortest path computation, referenced below, is a classical graph problem (seee, for example, Section 2.6 of [8]). The graphs that arise in the context of clock synchronization tend to be dense, making the Floyd-Warshall algorithm preferable over Johnson.

3 Clock Synchronization Algorithm

We define our system as composed of one or more threads of control, each thread a sequence of events. We are not concerned with the semantics of the events; for example, they might be communication notifications, system calls, or application stages.

Definition 1. *The* trace *is a tuple* $(E, \texttt{tid}, \leq, \texttt{ttime})$*, where:*

- E *is the set of events*
- $\texttt{tid} : E \to \mathbb{N}$ *is a function that matches each event with its thread id.*
- $\leq$ *is a happened-before relation over* E*:* $e_1 \leq e_2$ *implies the* e_1 *occured not later than* e_2*.*
- $\texttt{ttime} : E \to \mathbb{R}$ *is a function that matches each event to its thread-local timestamp. We assume that the clocks advance at the same rate for all the threads. We also assume, w.l.o.g., that the clocks are incrementing:*

$$\texttt{tid}(e_1) = \texttt{tid}(e_2), e_1 \leq e_2 \implies \texttt{ttime}(e_1) \leq \texttt{ttime}(e_2). \tag{1}$$

This definition of trace leaves a lot of freedom in implementing the tracing. The tracer may allocate a buffer per thread or use the same buffer for all threads and place a thread id field into the event record. Similarly, the order between events may be inferred from the events' semantics, as in [3], or from the event order in the shared buffer, as in [2].

Our goal is to define a global time function that is consistent with thread-local time (within a thread, time between events according to global time is the same as according to thread-local time) and the event order (an event's global time is no less than the global time of any of its predecessors):

Definition 2. *Given a trace* $(E, \texttt{tid}, \leq, \texttt{ttime})$*,* global time *is a function* $\texttt{gtime} : E \to \mathbb{R}$ *such that for every two events* $e_1, e_2 \in E$

$$\texttt{tid}(e_1) = \texttt{tid}(e_2) \implies \texttt{gtime}(e_1) - \texttt{gtime}(e_2) = \texttt{ttime}(e_1) - \texttt{ttime}(e_2) \tag{2}$$

$$e_1 \leq e_2 \implies \texttt{gtime}(e_1) \leq \texttt{gtime}(e_2) \tag{3}$$

Lemma 1. $\texttt{gtime}$ *is fully defined by a single value per thread,* $\{g_t\}_{t \in \texttt{tid}(E)}$*:*

$$\forall e \in E : \texttt{gtime}(e) = g_{\texttt{tid}(e)} + \texttt{ttime}(e). \tag{4}$$

Proof. Let $e_1, e_2 \in E$ and assume $\texttt{tid}(e_1) = \texttt{tid}(e_2) = t$. Let $g_t = \texttt{gtime}(e_1) - \texttt{ttime}(e_1)$. Then

$$\texttt{gtime}(e_2) = \texttt{gtime}(e_1) + \texttt{ttime}(e_2) - \texttt{ttime}(e_1) = g_t + \texttt{ttime}(e_2).$$

□

Example 1. In Fig. 1(a), if e_{21} is assigned global time 8, then, according to (2), $\texttt{gtime}(e_{22}) = 8 + (10 - 0) = 18$.

Next we note that any pair of events from different threads with a known order between them imposes a constraint on the difference of their respective g_ts.

Example 2. In the scenario depicted in Fig. 1(a), the event e_{11} is known to have preceded e_{21}. If we choose $g_1 \leq g_2$, this guarantees that $\texttt{gtime}(e_1) \leq \texttt{gtime}(e_2)$, i.e., (3) is satisfied with respect to e_{11} and e_{21}. If, on the other hand, we select $g_1 > g_2$, then $\texttt{gtime}(e_1) > \texttt{gtime}(e_2)$, contradicting the event order.

Lemma 2. *A function* $\texttt{gtime} : E \to \mathbb{R}$ *is a global time function if and only if it is of the form* (4) *and*

$$\forall e_1, e_2 \in E, e_1 \leq e_2 : g_{\texttt{tid}(e_1)} - g_{\texttt{tid}(e_2)} \leq \texttt{ttime}(e_2) - \texttt{ttime}(e_1). \tag{5}$$

Proof. According to Lemma 1, global time function must be of form (4), and (4) substituted into (3) gives (5). Conversely, (4) implies (2), and (4) together with (5) gives (3). □

Note that (5) defines a huge set of up to $|E|^2$ constraints, making it difficult to collect and use them. Our next step is to find a way to summarize these constraints.

Definition 3. *Let* e_1, e_2 *be two events. We say that* e_1 *is an* immediate predecessor *of* e_2*, denoted* $e_1 \prec e_2$*, if* e_1 *precedes* e_2 *and there is no event* e *between them:* $\nexists e : e_1 < e < e_2$*.*

Definition 4. *Let* $(E, \texttt{tid}, \leq, \texttt{ttime})$ *be a trace. For* $t_1, t_2 \in \texttt{tid}(E)$*, let*

$$L(t_1, t_2) = \{(e_1, e_2) | \texttt{tid}(e_1) = t_1, \texttt{tid}(e_2) = t_2, e_1 \prec e_2\}.$$

The constraints graph *for this trace is defined as the weighted clique*

$$G = (\texttt{tid}(E), \texttt{tid}(E) \times \texttt{tid}(E), w : \texttt{tid}(E) \times \texttt{tid}(E) \to \mathbb{R}),$$

$$w(t_1, t_2) = \min\{\texttt{ttime}(e_2) - \texttt{ttime}(e_1) | (e_1, e_2) \in L(t_1, t_2)\} \tag{6}$$

In other words, in the constraints graph, the weight of an edge (s, t) is the best bound on $g_s - g_t$ that we can derive using (5) from all event pairs related by $\prec$. We assume $\min(\emptyset) = \infty$.

Example 3. In the trace in Fig. 1(a), the order $\leq$ is linear, so G can be constructed in $O(|E|)$ steps. Figure 1(e) shows the constraints graph for this trace. Note that the constraints graph doesn't contain an explicit bound on the value of $g_2 - g_1$. However, it does contain an implicit constraint: $g_2 - g_1 = (g_2 - g_3) + (g_3 - g_1) \leq w(2, 3) + w(3, 1) = 10$. Similarly, while the explicit bound on $g_1 - g_2$ is $w(1, 2) = 0$, we also have $g_1 - g_2 = (g_1 - g_3) + (g_3 - g_2) \leq w(1, 3) + w(3, 2) = -6$.

To find the tightest bound on $g_s - g_t$, taking into account both explicit and implicit ones, we define

Definition 5. *Let* $(E, \texttt{tid}, \leq, \texttt{ttime})$ *be a trace and* $G = (\texttt{tid}(E), \texttt{tid}(E) \times \texttt{tid}(E), w)$ *the corresponding constraints graph. The* bounds graph *is a clique* $\bar{G} = (\texttt{tid}(E), \texttt{tid}(E) \times \texttt{tid}(E), \bar{w})$*, where* $\bar{w}(s,t)$ *is the weight of the shortest path between* s *and* t *in* G *(zero if* $s = t$*).*

The bounds graph can be computed using the Floyd-Warshall algorithm in $O(|\texttt{tid}(E)|^3)$. We now show that the bounds graph indeed summarizes (5).

Lemma 3. *A function* $\texttt{gtime} : E \to \mathbb{R}$ *is a global time function if and only if it satisfies* (4) *and*

$$\forall t_1, t_2 \in \texttt{tid}(E) : g_{t_1} - g_{t_2} \leq \bar{w}(t_1, t_2). \tag{7}$$

Proof. It suffices to show that (5) $\Leftrightarrow$ (7). Assume that (5) holds and (7) does not, i.e.,

$$\exists t, t' \in \texttt{tid}(E) : g_t - g_{t'} > \bar{w}(t, t'). \tag{8}$$

Since $\bar{w}(t,t')$ is defined as the weight of the shortest path between t and t' in G, there must exist a sequence of threads $t = t_0, t_1, \ldots, t_n = t'$ such that $\bar{w}(t,t') = w(t_0,t_1) + \ldots + w(t_{n-1}, t_n)$. According to (6), for every i, there exists a pair of events e_i, e_i' such that $\texttt{tid}(e_i) = t_i$, $\texttt{tid}(e_i') = t_{i+1}$, and $w(t_i, t_{i+1}) = \texttt{ttime}(e_i') - \texttt{ttime}(e_i)$. Substituting into (8),

$$g_t - g_{t'} > \bar{w}(t,t') = \sum_{i=0}^{n-1} w(t_i, t_{i+1}) = \sum_{i=0}^{n-1} (\texttt{ttime}(e_i') - \texttt{ttime}(e_i)) \overset{(5)}{\geq} \sum_{i=0}^{n-1} (g_{t_i} - g_{t_{i+1}}) = g_t - g_{t'}.$$

Conversely, assume that (7) holds, and let e, e' be any two events such that $e \leq e'$. Then there exists a sequence of events $e = e_0, e_1, \ldots, e_n = e'$ such that for any i, $e_i \prec e_{i+1}$. Let $t_i = \texttt{tid}(e_i)$. Then

$$g_{t_0} - g_{t_n} = \sum_{i=0}^{n-1} (g_i - g_{i+1}) \overset{(7)}{\leq} \sum_{i=0}^{n-1} \bar{w}(t_i, t_{i+1}) \leq \sum_{i=0}^{n-1} w(t_i, t_{i+1}) \leq$$
$$\leq \sum_{i=0}^{n-1} (\texttt{ttime}(e_{i+1}) - \texttt{ttime}(e_i)) = \texttt{ttime}(e') - \texttt{ttime}(e).$$

□

Example 4. Figure 1(f) shows the bounds graph corresponding to the trace in Fig. 1(a).

Having now summarized all the constraints imposed by the trace, we are ready to costruct some global time functions.

Lemma 4. *Let* $\tau \in \texttt{tid}(E)$ *be a thread, and let* α *be a parameter,* $0 \leq \alpha \leq 1$*. For every thread* t*, define*

$$g_t = \alpha \bar{w}(t, \tau) - (1 - \alpha) \bar{w}(\tau, t). \tag{9}$$

Then the function $\texttt{gtime}(e) = g_{\texttt{tid}(e)} + \texttt{ttime}(e)$ *is a global time function.*

Proof. We will use two obvious facts, both following from $\bar{w}$'s definition as the shortest path:

$$\forall i,j,k \in \texttt{tid}(E) \quad \bar{w}(i,j) \leq \bar{w}(i,k) + \bar{w}(k,j) \tag{10}$$

$$\forall i,j \in \texttt{tid}(E) \quad 0 = \bar{w}(i,i) \leq \bar{w}(i,j) + \bar{w}(j,i) \tag{11}$$

It suffices to show that for g_ts defined as in (9), (7) holds.

$$g_i - g_j \overset{(9)}{=} \alpha(\bar{w}(i,\tau) - \bar{w}(j,\tau)) - (1-\alpha)(\bar{w}(\tau,i) - \bar{w}(\tau,j)) \overset{(10)}{\leq}$$
$$\overset{(10)}{\leq} \alpha\bar{w}(i,j) - (1-\alpha)\bar{w}(j,i) \overset{(11)}{\leq} \alpha\bar{w}(i,j) + (1-\alpha)\bar{w}(i,j) = \bar{w}(i,j)$$

Note, by the way, that the inequalities above turn into equalities for $\alpha = 1, j = \tau$. Therefore the bounds imposed by $\bar{w}$ are tight. □

Example 5. The global time in Fig. 1(a) is obtained using $\tau = T_1, \alpha = \frac{1}{2}$.

Summing up, given a trace $(E, \texttt{tid}, \leq, \texttt{ttime})$, the algorithm performs the following steps:

- Build the constraints graph. The complexity of this step depends on $\leq$'s definition; for a linear order it's $O(|E|)$.
- Build the bounds graph. This can be done in $O(|\texttt{tid}(E)|^3)$.
- Choose τ and α and build `gtime` according to (9). This takes $O(|\texttt{tid}(E)|)$ steps.

4 Implementation Aspects

In this section, we discuss the application of the clock synchronization algorithm to traces collected on Cell; the issues we encountered; and the solutions that were adopted.

The PDT [2] events correspond to method calls in PDT-instrumented libraries. Such instrumentation is available, for example, for libraries that handle communications between the cores via mailboxes and communication between the cores and main memory through the DMA. In addition, certain events are monitored to provide enough information for trace post-processing. In particular, PDT traces all the context switch events on the SPEs.

Events monitored by the PDT can rarely be used to establish inter-event order based on the event semantics, as in [3]. To improve the precision of time computation, the PDT forces global ordering of all traced events. All the events are written into a single buffer in the order of their arrival.

Context switches pose a problem to the algorithm because the decrementer register value is restored when a thread resumes. Our solution was to treat each live interval (a thread execution from one context switch to the next) as a "thread" in terms of the clock synchronization algorithm. This solution has the drawback of less events (and hence less constraints) generated for each "thread",

Table 1. Clock synchronization precision by benchmark

Name	Max error	Max error (SPEs)	Av. error	Av. error (SPEs)
BlackScholes	63	63	39	33
FFT16M	24	18	20	12
JuliaSet	34	19	27	12
Matrix_mul	33	11	31	8

but it still gives better precision than the alternatives. However, the PPE threads are all treated as one, since they use the same clock.

Another issue is possible imprecision in collecting timestamps and event order information. For example, if the timestamp precision is $\pm\delta$, then constraint (3) becomes less tight:

$$g_{\mathtt{tid}(e_1)} + \mathtt{ttime}(e_1) - \delta \leq g_{\mathtt{tid}(e2)} + \mathtt{ttime}(e_2) + \delta. \tag{12}$$

In most practical situations, however, the value of δ is not known and cannot be taken into account when the constraints are computed. Consequently, constraints graph may contain negative-weight cycles of weight up to -2δ, and the all-pairs shortest path computation would fail.

One solution to this problem lies with the tracer, which can take measures to reduce the δ. In particular, the PDT event timestamp corresponds to the time event was written into the buffer rather than when it occurred. This difference is very small, but significant for the algorithm, since it prevents negative-weight cycles. Once we adopted this approach, we no longer saw negative cycles in any PDT-generated traces since this approach was adopted. Another solution is based on the observation that if a negative cycle of weight $-d$ is discovered, then $\delta \geq d/2$, and all the constraints should be made less tight as in (12). This can be done efficiently by setting $w(i,j) \leftarrow w(i,j) + \frac{d}{2}$. This step can be repeated until no negative cycles remain, generating in the process an estimate for δ.

5 Experimental Results

To estimate the practical applicability of the clock synchronization algorithm, we ran it on traces generated by several workloads from Cell SDK 3.0 [9]: BlackScholes, FFT16M, JuliaSet and Matrix_Mul. All the workloads executed on a IBM QS20 BladeCenter running two Cell BE Processors at 3.2GHz, under Fedora 7 Linux. The PDT was configured to trace all stalling events, such as waiting for a mailbox or DMA transfer to arrive. Non-stalling events, such as asynchronously issuing a DMA request, were not traced. The workloads were configured to utilize all the 16 SPUs on the blade.

Let s and t be two threads. According to (7), $-\bar{w}(t,s) \leq g_s - g_t \leq \bar{w}(s,t)$. Thus $\bar{w}(s,t)+\bar{w}(t,s)$ is an upper bound on the possible shift of the time scales of s and t for any global time on a trace. Table 1 aggregates these timing error bounds, measured in ticks, over trace pairs. The first column shows the maximum timing

error for all thread pairs; the second column lists the maximum error over all SPE-SPE thread pairs; the third column contains the average timing error for SPE-PPE thread pairs; and the fourth column shows the average timing error for SPE-SPE thread pairs.

As Table 1 shows, the algorithm generated high-precision results for these benchmarks, with the max error ranging from 63 ticks for BlackScholes to 33 ticks for Matrix_mul, and the average error about two thirds of that. For a 14.8 MHz clock, this translates into 1.4–2.8 μs average error bounds. For all those benchmarks, most of the work is done on the SPEs, with the PPE largely responsible for intialization and coordination of the SPE threads. Since the events on the SPEs are much more dense, it is not surprising that the errors in the relative timing of SPE events are typically smaller. The most dramatic drop is observed on Matrix_mul, where the error is limited by 33 for all the events, but falls to 11 when only SPU events are considered. Given this pattern, we recommend selecting the global time function defined by (9) in which τ is the PPE "thread" and $\alpha = \frac{1}{2}$, cutting the SPE-PPE error estimate by half.

We also conducted an experiment to check how the instrumentation level affects clock synchronization precision, by fixing a benchmark (FFT16M) and running it with different levels of instrumentation. The full instrumentation, which traces all (stalling and non-stalling) communications, naturally gave the best precision (maximal error bound of 20 ticks, 11 ticks if restricted to SPE events). The basic instrumentation, which only traces thread start and end for SPE threads, gave the worst precision, 17,590 ticks. For a more interesting example of algorithm behavior on sparse traces, we configured the PDT to trace only stalls occurring in FFT's outer loop. This generated 88 events per SPE, with the average of 1,993,360 ticks between events. However, the maximal error bound on this trace was 53 ticks. This result can be understood if we take into account that this instrumentation level generates bursts of several temporally close events per thread, and that the bursts come at approximately the same time across threads, due to phase synchronization.

6 Conclusions

The absence of wall-clock timestamps in the trace is a severe limitation on the trace visualization and usage. In this paper, we showed how this problem can be solved during trace post-processing. We presented an algorithm for estimating the wall-clock time based on thread-local time and partial event ordering. The algorithm is linear in the size of the partial order relation between the events, and cubical in the number of threads. We have shown how to estimate the precision that the algorithm provides on a particular trace, and how it can recover from minor imprecision in time measurements and event ordering. The algorithm was used on several workloads from the Cell SDK 3.0, traced with different instrumentation levels, and showed good precision results. It's significant to note that the algorithm is by definition precise enough to preserve important properties of the input trace, namely, intra-thread timing and inter-thread event

order. The algorithm is used in the Trace Analyzer tool, publicly available as part of the Visual Performance Analyzer [10]. We also apply it to traces from other multicores.

There are several important directions for future research. Our algorithm does not handle the clock drift. Since the algorithm depends on the distribution of events in the trace, it may be possible to improve the precision of the algorithm on sparse traces by allowing the tracer to generate a small number of extra events on each core; how to achieve this in an optimal way remains a topic for further research. Another requirement that the time synchronization algorithm imposes on the tracer tool is the preservation of enough event order data. Currently the PDT achieves this by sharing the same buffer for all the events. Using different buffers for different cores or accumulating several events on core before flushing them to the global buffer can reduce the tracing overhead. It remains to be seen how to optimally balance overhead reduction with clock precision.

Acknowledgements

We thank Ayal Zaks, Bilha Mendelson, Javier Turek, Orit Edelstein, Uzi Shvadron, and Yehuda Naveh for the many interesting discussions of the topic. We are also very grateful to the referees for the helpful suggestions.

References

1. Chen, T., Raghavan, R., Dale, J., Iwata, E.: Cell Broadband Engine architecture and its first implementation, `http://www.ibm.com/developerworks/power/library/pa-cellperf/`
2. Biberstein, M., Chang, M.S., Mendelson, B., Shvadron, U., Turek, J.: Trace-based performance analysis on Cell BE. In: Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) (2008)
3. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (1978)
4. Mattern, F.: Virtual time and global states of distributed systems. In: Parallel and Distributed Algorithms: Proceedings of the International Workshop on Parallel and Distributed Algorithms (1988)
5. Williams, C., Reynolds, P.F., de Supinski, B.R.: Delta coherence protocols. IEEE Concurrency 8(3), 23–29 (2000)
6. Mills, D.L.: Internet time synchronization: The network time protocol. In: Yang, Z., Marsland, T.A. (eds.) Global States and Time in Distributed Systems. IEEE Computer Society Press, Los Alamitos (1994)
7. Maillet, E., Tron, C.: On efficiently implementing global time for performance evaluation on multiprocessor systems. Journal of Parallel and Distributed Computing 28(1), 84–93 (1995)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
9. IBM: Cell BE SDK 3.0, `http://-www.ibm.com/developerworks/power/cellpkgdownloads.html`
10. IBM: Visual Performance Analyzer, `http://www.alphaworks.ibm.com/tech/vpa`