Fast Min-plus Convolution and Deconvolution on GPUs

Natchanon Luangsomboon
University of Toronto
Toronto, Ontario
nat.luangsomboon@mail.utoronto.ca

Robert Hesse
University of Toronto
Toronto, Ontario
hesserob@eecg.toronto.edu

Jörg Liebeherr
University of Toronto
Toronto, Ontario
jorg@ece.utoronto.ca

ABSTRACT

The min-plus convolution and deconvolution operations are frequently needed in the network calculus for computing performance metrics. However, due to their computational complexity, the operations can become impractical with large data sets. We provide a GPU-accelerated implementation that achieves a 400x speedup for a data set with a half million data points, reducing the computation time from more than an hour to about 10 seconds. We also determine the maximum ranges for which the convolution and deconvolution of data traces finite support can be correctly computed.

CCS CONCEPTS

• **Mathematics of computing** → **Solvers**; • **Networks** → **Network performance analysis**;

KEYWORDS

Network calculus, Min-plus convolution, GPU implementation

ACM Reference Format:
Natchanon Luangsomboon, Robert Hesse, and Jörg Liebeherr. 2017. Fast Min-plus Convolution and Deconvolution on GPUs. In *Proceedings of 11th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2017).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3150928.3150958

1 INTRODUCTION

This paper addresses numerical issues that arise in computations in the network calculus [13]. The network calculus is a framework for the analysis of communication networks that has been applied to avionic networks [5], automobile networks [7, 8], network-on-chip systems [15], energy storage in smart grids [6, 19], and map-reduce computations [4]. In the min-plus network calculus, arrivals and departures at a network element (see Figure 1) are represented as functions of time, where $A(t)$ and $D(t)$ describe the cumulative amount of arriving or departing traffic until time t. Here, a network element can be a packet switch, a buffered link, a network cable, or any combination thereof. The available service in a time interval of length τ is expressed by a function $S(\tau)$. The functions describing arrivals, departures and available service are non-decreasing and non-negative, and, by convention, these functions are set to zero for values of $t \leq 0$. The set of functions of this type is frequently denoted by $\mathcal{F}_o$. There are discrete-time and continuous-time formulations of the network calculus. Since this paper is concerned with computational aspects, we will consider a discrete-time setting with $t \in \mathbb{Z}$.

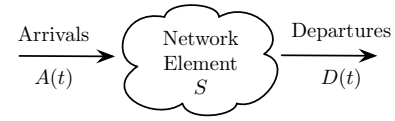


Figure 1: Network element with arrivals and departures.

The min-plus network calculus defines two operations, the min-plus convolution and deconvolution, that play an important role in the characterization and analysis of a network. The min-plus convolution for two functions F and G is defined as

$$F \otimes G(t) = \inf_{s \in \mathbb{Z}} \{F(s) + G(t-s)\},$$

and the min-plus deconvolution is given by

$$F \oslash G(t) = \sup_{s \in \mathbb{Z}} \{F(t+s) - G(s)\}.$$

If $F, G \in \mathcal{F}_o$, the operations simplify to

$$\begin{aligned} F \otimes G(t) &= \min_{0 \leq s \leq t} \{F(s) + G(t-s)\}, \\ F \oslash G(t) &= \max_{s \geq 0} \{F(t+s) - G(s)\}. \end{aligned} \tag{1}$$

This paper is concerned with the numerical computation of these operations.

As we will briefly discuss in Section 2, the min-plus convolution operator enables a concise description of service guarantees at a network element or a sequence of several network elements. The min-plus deconvolution operator is used to express bounds on backlog, delay, and output burstiness, as well as time-invariant descriptions of traffic. From (1), we see that the computational complexity of the convolution and deconvolution grows quadratically in t, and hence, may result in long computation times when the data set is large. Moreover, computing the deconvolution $F \oslash G$ is not entirely straightforward when the function values of F and G are only available for a finite set of time values, e.g., $t = 0, 1, \ldots, T$, since a correct computation of the deconvolution $F \oslash G(T)$ requires $F(t)$ and $G(t)$ for values $t > T$.

For concave functions F and G, the min-plus convolution simplifies to $F \otimes G(t) = \min\{F(t), G(t)\}$. This has been exploited in existing tools for network calculus analysis [1, 2, 18] of piecewise linear functions. Without an approximation by piecewise linear functions, the convolution and deconvolution of general functions is known to execute slowly on large data sets. In this paper, we show how to accelerate the computations with a GPU implementation of the min-plus convolution and deconvolution. Our tool is developed

VALUETOOLS 2017, December 5–7, 2017, Venice, Italy

ACM ISBN 978-1-4503-6346-4/17/12.
https://doi.org/10.1145/3150928.3150958

in the NVIDIA CUDA language for general-purpose computing on Graphical Processing Units (GPUs) [16, 17]. GPUs have hundreds (sometimes thousands) of computing cores and are highly suitable for compute-intensive applications with a high degree of parallelism. Originally conceived for the acceleration of computer displays, GPUs have led to breakthroughs in many fields, such as data analytics, deep learning, and fluid simulations. Compared to a generic MATLAB computation [14], our implementation reduces the computation time of the convolution and deconvolution for functions with 500 000 data points by a factor of more than 400, from more than an hour to 10 seconds. Even for close to 1 million data points, the GPU acceleration can compute the convolution in less than one minute. Our software, referred to as *turboNC* [10], can be adapted and integrated into existing tools to reduce computation times.

The contributions of this paper are as follows. We present a fast computation of the min-plus convolution and deconvolution on GPUs. We provide a comparative evaluation of the computation times as a function of the length of a data trace. Furthermore, extending results in [11, 12], we address how to correctly compute $F \otimes G$ and $F \oslash G$ when the function values of F and G are only available for a limited range.

The rest of the paper is structured as follows. In Section 2, we briefly explain the role of the convolution and deconvolution in the min-plus network calculus. In Section 3, we show how to achieve a correct computation of the convolution and deconvolution of functions when function values are limited to a given range. In Section 4, we discuss our CUDA implementation. In Section 5, we evaluate the computation times of the CUDA implementation. We briefly conclude in Section 6.

2 ROLES OF THE (DE-) CONVOLUTION

The min-plus convolution and deconvolution operations in the network calculus play an important role in computing traffic envelopes and performance bounds. Given a network element as shown in Figure 1 with a cumulative arrival function A and departure function D with $A, D \in \mathcal{F}_o$, a function $S \in \mathcal{F}_o$ is called an *(exact) service curve* of the network element if, for all $t \in \mathbb{R}$, $D(t) = A \otimes S(t)$. If the network element is a work-conserving link with rate C, it has an exact service curve $S(t) = \max\{Ct, 0\}$. A network element that delays arriving traffic by d time units has the exact service curve $S(t) = 0$ if $t \leq d$, and $S(t) = \infty$ if $t > d$. Functions that satisfy the inequality $D(t) \geq A \otimes S(t)$ for all $t \in \mathbb{R}$ are called *lower service curves*. Lower service curves express lower bounds on the offered service at a network element. The convolution operation is also used to express the available service on a network path. Given a sequence of network elements, with service curves $S_1, \ldots, S_N$, the convolution $S_1 \otimes \ldots \otimes S_N$ is a service curve for the entire sequence of network elements.

For an arrival function A, a function $E \in \mathcal{F}_o$ that provides a time-invariant bound on A in the sense that

$$E(\tau) \geq A(t + \tau) - A(t)\,, \quad \forall t, \tau \geq 0\,,$$

is called a *traffic envelope* for A. We write $A \sim E$ to indicate that E is a traffic envelope for A. The property $A \sim E$ can be equivalently expressed in terms of the deconvolution by $E(\tau) \geq A \oslash A(\tau)$. The smallest possible envelope for an arrival function A, referred to as the *empirical envelope* of A [9], is therefore given by $A \oslash A$.

Given a network element with lower service curve S, and an arrival function A with traffic envelope E, i.e., $A \sim E$, the min-plus deconvolution of $E \oslash S$ yields performance bounds at the network element. The function $E \oslash S(\tau)$ is a min-plus traffic envelope for the departure function D for $\tau > 0$, that is, $D \sim E \oslash S$. The backlog for arbitrary times $t > 0$ is bounded by $E \oslash S(0)$. Finally, as long as traffic is served in the order of arrivals, the delay at all times $t > 0$ is bounded by the smallest number $d \geq 0$ that satisfies $E \oslash S(-d) \leq 0$. Note that the computation of delay bounds evaluates the deconvolution $E \oslash S$ for negative arguments.

3 CONVOLUTIONS AND DECONVOLUTIONS OF FUNCTIONS AND DATA TRACES

This section addresses questions that arise in the numerical computation of the convolution and deconvolution for data traces and functions that are available only for a finite time interval. Suppose that we want to compute the deconvolution $F \oslash G$ for two functions, that are provided as data traces. That is, function $F(t)$ is available for the interval $T_{\min,F} \leq t \leq T_{\max,F}$ and $G(t)$ is available for the interval $T_{\min,G} \leq t \leq T_{\max,G}$. Then the question arises for which time interval we can correctly compute the convolution or deconvolution? To see that this is a non-trivial question, note that the deconvolution expression in (1) requires function values for $t > \max\{T_{\max,F}, T_{\max,G}\}$. In this section, we identify the ranges for which the convolution and deconvolution can be computed.

3.1 Extrapolations

Consider the construction of a cumulative arrival function from a trace, e.g., a data trace of captured packet sizes on a network link or a video trace consisting of the frame sizes of a compressed video sequence. Data of a trace is provided in the form of a time series $(t_1, y_1), (t_2, y_2), ..., (t_n, y_n)$ with $t_i < t_j$, if $i < j$, where the tuple (t_i, y_i) describes that an amount of data y_i is associated with time t_i. A cumulative function F is constructed from the trace by

$$F(t) = \sum_{t_i < t} y_i\,,$$

Since

$$F(t) = 0\,, \qquad \forall t \leq t_1\,,$$
$$F(t) = F(t_n)\,, \qquad \forall t \geq t_n\,,$$

we can extrapolate the values of the cumulative function F to all values $t \in \mathbb{Z}$. We can view the function F as having two 'plateaus': a plateau at zero for values $t \leq t_1$ and a plateau at $F(t_n)$ for values $t \geq t_n$.

A generalization of the extrapolation to an arbitrary, that is, not necessarily cumulative, function, defined for a finite time interval $T_{\min} \leq t \leq T_{\max}$ leads to

$$F(t) = \begin{cases} F(T_{\min})\,, & t < T_{\min}\,, \\ F(t)\,, & T_{\min} \leq t \leq T_{\max}\,, \\ F(T_{\max})\,, & T_{\max} < t\,. \end{cases} \tag{2}$$

We say that a function F has a 'left plateau at T', if $F(t) = F(T)$ for all $t \leq T$. We say that F has a 'right plateau at T', if $F(t) = F(T)$ for

all $t \geq T$. The purpose of the plateau extrapolation is to capture a function whose support is known only for a limited range.

The concept of extrapolation is also useful for vector representations of functions $F \in \mathcal{F}_o$, e.g., $F(t) = \sqrt{t}$ for $t > 0$. With tools such as MATLAB, function values are stored in vectors with a fixed length, and each vector element is assigned a function value on a discrete-time scale, e.g., the value of $\sqrt{5}$ may be stored in the 5th field of the vector. Since function values are computed only up to the length of the vector, the stored function reflects F only up to a maximum time value, which we denote by $T_{\max,F}$. To distinguish the original function F from the stored function, we define $F_{[T_{\max,F}]}$ using extrapolation as

$$F_{[T_{\max,F}]}(t) = \begin{cases} 0\,, & t \leq 0\,, \\ F(t)\,, & 0 < t \leq T_{\max,F}\,, \\ F(T_{\max,F})\,, & t > T_{\max,F}\,. \end{cases}$$

We can think of F as the actual function and $F_{[T_{\max,F}]}$ as its vector representation in a data structure.

3.2 Computations for functions in $\mathcal{F}_o$

Our first result, which corresponds to [12, Lemma 1], is on the convolution of two functions F and G in $\mathcal{F}_o$ whose values are available from zero until $T_{\max,F}$ and $T_{\max,G}$, respectively.

Lemma 3.1. *If $F, G \in \mathcal{F}_0$, then, for all $t \leq \min\{T_{\max,F}, T_{\max,G}\}$,*

$$F_{[T_{\max,F}]} \otimes G_{[T_{\max,G}]}(t) = F \otimes G(t)\,.$$

Thus, if the function values of each function are available up to a time limit, then the convolution can be computed up to the smaller of the time limits.

Proof. Since $F_{[T_{\max,F}]}, G_{[T_{\max,G}]} \in \mathcal{F}_0$, we have that each $t \leq \min\{T_{\max,F}, T_{\max,G}\}$ satisfies

$$\begin{aligned} F_{[T_{\max,F}]} \otimes G_{[T_{\max,G}]}(t) &= \min_{0 \leq s \leq t} \{F_{[T_{\max,F}]}(s) + G_{[T_{\max,G}]}(t-s)\} \\ &= F \otimes G(t)\,. \end{aligned}$$

□

For the deconvolution of two functions in $\mathcal{F}_o$, the constraints are more involved, since the deconvolution of two functions is defined as the maximum difference of a time-shifted version of the first function and the second function over an unlimited range. We can provide a range where the deconvolution is valid, as long as there is a last time when the two functions intersect. Specifically, we define $\tau_t(F, G)$ to denote

$$\tau_t(F, G) = \sup\{s \mid F(s+t) \geq G(s)\}\,,$$

that is, $\tau(F, G)$ denotes the last intersection of a right-shifted version of F with G. The following result extends [12, Theorem 1] by relaxing assumptions on F and G.

Lemma 3.2. *Let $F, G \in \mathcal{F}_o$ and let $T_{\max,F}, T_{\max,G}$ and $\tau_t(F, G)$ be as defined above. Then,*

$$F_{[T_{\max,F}]} \oslash G_{[T_{\max,G}]}(t) = F \oslash G(t)\,,$$

if $T_{\max,G} \geq \tau_t(F, G)$ and $T_{\max,F} \geq t + T_{\max,G}$.

To compute $F \oslash G(t)$ correctly, we hence have two constraints on the time limits of F and G. First, the values of G must be available up to the last intersection of a left-shifted function F and G. Second, the values of F must be available for t time units more than the time limit of G.

Proof. Fix a value of t. Since, by definition of $\tau_t(F, G)$,

$$\sup_{s > \tau_t(F,G)} \{F(t+s) - G(s)\} < 0\,,$$

and since $F, G \in \mathcal{F}_o$ implies $F(t) - G(0) \geq 0$, and with $T_{\max,G} \geq \tau_t(F, G)$, we have

$$\begin{aligned} \sup_{s > T_{\max,G}} \{F(t+s) - G(s)\} &\leq \sup_{s > \tau_t(F,G)} \{F(t+s) - G(s)\} \\ &\leq \sup_{s \leq \tau_t(F,G)} \{F(t+s) - G(s)\} \\ &\leq \sup_{s \leq T_{\max,G}} \{F(t+s) - G(s)\}\,. \end{aligned}$$

Thus,

$$\begin{aligned} F \oslash G(t) &= \max \left\{ \begin{array}{l} \sup_{s \leq T_{\max,G}} \{F(t+s) - G(s)\}\,, \\ \sup_{s > T_{\max,G}} \{F(t+s) - G(s)\} \end{array} \right\} \\ &= \sup_{s \leq T_{\max,G}} \{F(t+s) - G(s)\} \\ &= \max \left\{ \begin{array}{l} \sup_{s \leq T_{\max,G}} \{F(t+s) - G(s)\}\,, \\ \sup_{T_{\max,G} < s \leq T_{\max,F} - t} \{F(t+s) - G(T_{\max,G})\}\,, \\ \sup_{T_{\max,F} - t < s} \{F(T_{\max,F}) - G(T_{\max,G})\} \end{array} \right\} \\ &= F_{[T_{\max,F}]} \oslash G_{[T_{\max,G}]}(t)\,. \end{aligned}$$

□

3.3 Computations for data traces

When convolving or deconvolving data traces, the function F representing the data trace has the extrapolated form given in (2), with a left and a right plateau. The following lemma states that the convolution of such functions also has left and right plateaus.

Lemma 3.3. *Let F, G be arbitrary single-variable functions with left and right plateaus at $T_{\min,F}, T_{\max,F}$ and $T_{\min,G}, T_{\max,G}$, respectively. Then $F \otimes G$ has left and right plateaus at $T_{\min,F} + T_{\min,G}$ and $T_{\max,F} + T_{\max,G}$.*

Note that, different from assumptions in [11, 12], we do not require $F, G \in \mathcal{F}_o$. In particular, the functions need not be non-decreasing, and may have negative values. A consequence of the theorem is that the convolution must only be computed for values of t with $T_{\min,F} + T_{\min,G} \leq t \leq T_{\max,F} + T_{\max,G}$.

Proof. First consider that $t < T_{\min,F} + T_{\min,G}$. We obtain

$$\begin{aligned} F \otimes G(t) &= \inf_{s \in \mathbb{Z}} \{F(s) + G(t-s)\} \\ &= \min \left\{ \begin{array}{ll} \inf_{s < t - T_{\min,G}} & \{F(T_{\min,F}) + G(t-s)\}\,, \\ \min_{t - T_{\min,G} \leq s \leq T_{\min,F}} & \{F(T_{\min,F}) + G(T_{\min,G})\}\,, \\ \inf_{T_{\min,F} < s} & \{F(s) + G(T_{\min,G})\} \end{array} \right\} \\ &= \min \left\{ \begin{array}{ll} \inf_{s < T_{\min,F}} & \{F(s) + G(T_{\min,F} + T_{\min,G} - s)\}\,, \\ & F(T_{\min,F}) + G(T_{\min,G})\,, \\ \inf_{T_{\min,F} < s} & \{F(s) + G(T_{\min,F} + T_{\min,G} - s)\} \end{array} \right\} \\ &= \inf_{s \in \mathbb{Z}} \{F(s) + G(T_{\min,F} + T_{\min,G} - s)\} \\ &= F \otimes G(T_{\min,F} + T_{\min,G})\,. \end{aligned}$$

Similarly, for $t > T_{\max,F} + T_{\max,G}$ we get

$$F \otimes G(t) = \min \left\{ \begin{array}{c} \inf\limits_{s<T_{\max,F}} \{F(s) + G(T_{\max,G})\}, \\ \inf\limits_{T_{\max,F} \le s \le t - T_{\min,G}} \{F(T_{\max,F}) + G(T_{\max,G})\}, \\ \inf\limits_{s<t-T_{\min,G}} \{F(T_{\max,F}) + G(t-s)\} \end{array} \right\} = F \otimes G(T_{\max,F} + T_{\max,G}) .$$

□

The computation of $F \otimes G$ in the range given by Theorem 3.3 follows from the next theorem.

Theorem 3.4. *Let F, G be arbitrary single-variable functions with left and right plateaus $T_{\min,F}, T_{\max,F}$ and $T_{\min,G}, T_{\max,G}$, respectively, and define*

$$l = \max\{T_{\min,F}, t - T_{\max,G}\},$$
$$r = \min\{T_{\max,F}, t - T_{\min,G}\},$$

Also define the following modified convolution operations:

$$F \otimes_{l,r} G(t) = \min_{l \le s \le r} \{F(s) + G(t-s)\},$$

$$F \dot{\otimes} G(t) = \begin{cases} F(T_{\min,F}) + \inf\limits_{t-T_{\min,F}<s} \{G(s)\}, & t < T_{\min,F} + T_{\max,G}, \\ \inf\limits_{s<t-T_{\max,G}} \{F(s)\} + G(T_{\max,G}), & t \ge T_{\min,F} + T_{\max,G}, \end{cases}$$

$$F \ddot{\otimes} G(t) = \begin{cases} \inf\limits_{s>t-T_{\min,G}} \{F(s)\} + G(T_{\min,G}), & t < T_{\max,F} + T_{\min,G}, \\ F(T_{\max,F}) + \inf\limits_{s<t-T_{\max,F}} \{G(s)\}, & t \ge T_{\max,F} + T_{\min,G}. \end{cases}$$

Then, for each $t \in [T_{\min,F} + T_{\min,G}, T_{\max,F} + T_{\max,G}]$, it holds that

$$F \otimes G(t) = \min\{F \otimes_{l,r} G(t), F \dot{\otimes} G(t), F \ddot{\otimes} G(t)\}.$$

Proof. Consider a $t \in [T_{\min,F} + T_{\min,G}, T_{\max,F} + T_{\max,G}]$. Then $l \le r$ and we have

$$\inf_{s<l}\{F(s) + G(t-s)\} = \inf_{s<\max\{T_{\min,F}, t-T_{\max,G}\}} \{F(s) + G(t-s)\}$$

$$= \begin{cases} \inf\limits_{s<T_{\min,F}} \{F(T_{\min,F}) + G(t-s)\}, & T_{\min,F} > t - T_{\max,G}, \\ \inf\limits_{s<t-T_{\max,G}} \{F(s) + G(T_{\max,G})\}, & T_{\min,F} \ge t - T_{\max,G}, \end{cases}$$

$$= \begin{cases} F(T_{\min,F}) + \inf_{t-T_{\min,F}<s}\{G(s)\}, & t < T_{\min,F} + T_{\max,G}, \\ \inf_{s<t-T_{\max,G}}\{F(s)\} + G(T_{\max,G}), & t \ge T_{\min,F} + T_{\max,G}, \end{cases}$$

$$= F \dot{\otimes} G(t).$$

Similarly,

$$\inf_{s>r}\{F(s) + G(t-s)\} = \inf_{s>\min\{T_{\max,F}, t-T_{\min,G}\}} \{F(s) + G(t-s)\}$$

$$= \begin{cases} \inf\limits_{s>t-T_{\min,G}} \{F(s) + G(T_{\min,G})\}, & T_{\max,F} > t - T_{\min,G}, \\ \inf\limits_{s>T_{\max,F}} \{F(T_{\max,F}) + G(t-s)\}, & T_{\max,F} \le t - T_{\min,G}, \end{cases}$$

$$= F \ddot{\otimes} G(t).$$

Thus,

$$F \otimes G(t) = \min \left\{ \begin{array}{c} \inf\limits_{s<l}\{F(s) + G(t-s)\}, \\ \min\limits_{l \le s \le r} \{F(s) + G(t-s)\}, \\ \inf\limits_{r<s}\{F(s) + G(t-s)\} \end{array} \right\} = \min\{F \otimes_{l,r} G(t), F \dot{\otimes} G(t), F \ddot{\otimes} G(t)\}.$$

□

With Theorem 3.4, we have a computational method for the convolution for arbitrary functions with left and right plateaus. We can use the theorem to compute a deconvolution. This is our final result.

Theorem 3.5. *Let F, G be arbitrary single-varied functions, and define*

$$F'(t) = -F(t) \quad \text{and} \quad G'(t) = G(-t).$$

Then

$$F \oslash G(t) = -F' \otimes G'(t).$$

Note that if F and G have left and right plateaus, so do F' and G'.

Proof. Consider

$$F \oslash G(t) = \sup_{s \in \mathbb{Z}}\{F(t+s) - G(s)\} = -\inf_{s \in \mathbb{Z}}\{-F(s) + G(-(t-s))\} = -\inf_{s \in \mathbb{Z}}\{F'(s) + G'(t-s)\} = -F' \otimes G'(t).$$

□

4 CUDA IMPLEMENTATION

We have developed a GPU-accelerated computation of the min-plus convolution and deconvolution for arbitrary data traces. As GPU architecture, we selected NVIDIA CUDA [16]. The algorithms from the previous section are evaluated on a Tesla C2050 computing processor with 448 cores. The processor is installed on a Linux desktop equipped with an Intel Xeon 2.13 GHz CPU. By avoiding any architecture specific code, our implementation can be run on any CUDA-enabled device.

4.1 General-purpose computing on graphics processing units

General-purpose computing on graphics processing units (GPGPU) is a technique for using GPUs to perform computations in applications that are traditionally handled by a CPU. Since GPUs greatly outperform CPUs in arithmetic throughput and memory bandwidth, they are ideal for applications that are compute-intensive and with opportunities for parallel computations. The most common performance limitation in modern computing is that on-chip data cannot be served to the processors of a chip fast enough. CPUs try to reduce the effects of large on-chip memory latencies by exploiting data locality with large on-chip caches and instruction-level parallelism. GPUs take a different approach to the same problem in that they hide memory latency with calculations. That is, while some tasks are waiting for data to continue their computation, other tasks are computed at the same time. The key to reach high performance in

```
__global__ void convolve(
    const double *const F, const int lenF,
    const double *const G, const int lenG,
    double *const FG, const int len) {

    const int t = blockDim.x * blockIdx.x + threadIdx.x;
    if (t >= len) return;

    double result = INFINITY;
    const int r = min(t+1, lenG), l = max(0, t-lenF+1);
    for (int s = l; s < r; s++)
        result = fmin(result, F[t-s] + G[s]);
    FG[t] = result;
}
```

Figure 2: CUDA code segment for $F \otimes G$.

```
// Invoke kernel
const int threadsPerBlock = 256;
const int blocks = (len + threadsPerBlock - 1) / threadsPerBlock;
convolve<<<blocks, threadsPerBlock>>>(F, N_F, G, N_G, FG, N_FG);
```

Figure 3: CUDA kernel in C.

GPGPU is to keep all processors of the GPU busy at all times by providing enough computational tasks with a high ratio of arithmetic operations to memory operations.

CUDA GPUs consist of several *streaming multiprocessors* (SM), which, in turn, consist of several scalar CUDA cores. The Tesla C2050 processor has 16 SMs. The basic unit of work in terms of CUDA is a *thread*, which is executed on a single CUDA core. Threads are defined as kernel functions in C and work on a specified subset of data. A CUDA core can perform a context switch between threads with negligible overhead. This allows the core to hide memory latency by switching between the contexts of threads that are ready for execution. Multiple threads are combined to a common workgroup called a *block*, which is executed on the same SM. A block is the basic unit of scheduling in CUDA. Once all threads of a block have finished execution, the next block is scheduled for execution on the same SM. Several blocks create a grid, which represents the entire computational task executed on the GPU.

4.2 Mapping to CUDA Architecture

The min-plus convolution and deconvolution are a good fit for an independent execution by parallel threads, since the result for each value of t can be computed independently. In our implementation, we have mapped the computation for each value t to a separate thread. This mapping allows the computation of the optimization (minimum or maximum) for each t in parallel.

Consider the convolution of two functions $F, G \in \mathcal{F}_o$ which are specified for $t = 0, 1, \ldots,$ lenF-1 and $t = 0, 1, \ldots,$ lenG-1. We only concern ourselves with the modified convolution $\otimes_{l,r}$, as it is the only computation from Section 3 with quadratic time-complexity and will benefit most from parallelism. The other modified convolutions ($\dot{\otimes}, \tilde{\otimes}$) are of linear time-complexity and are implemented outside of CUDA.

The convolution operation is divided into N subproblems, whereby the n-th thread computes

$$F \otimes_{l,r} G(n-1) = \min_{s=\max(0,\, t-N+1),\ldots,\min(n-1,\, t)} \{F(s) + G(n-1-s)\}\,.$$

The CUDA code for this computation is shown in Figure 2. Mapping the function values to threads in this fashion may appear naive, but it leads to an efficient use of the CUDA architecture. The computation in Figure 2 is then formulated as a CUDA kernel in the C programming language. The relevant code segment is shown in Figure 3. Note that we set the number of threads per block to 256. This code is compiled for execution on NVIDIA GPUs.

The partitioning of a task into blocks and threads are design parameters, which need to be carefully selected. The block size determines the number of threads running parallel on the same SM. Since all threads on an SM share a register file, the block size must be set so that the total number of registers used by the threads does not exceed the size of the register file. This may reduce the number of concurrent threads, which, in turn, may expose the memory latency of other threads. When designing a CUDA application, these and many other trade-offs need to be carefully considered to achieve maximum performance from a given architecture.

The full CUDA implementation requires additional wrapper functions for memory allocation of data structures, data transfer to and from GPU memory and thread synchronization with the CPU. In total, our implementation has approximately 1500 lines of code.

The CUDA C code of the min-plus convolution and deconvolution can be run without any additional software packages. For ease of use, we have compiled the implementation into a MATLAB executable file that can be called from within MATLAB. We point out that there are several toolboxes available for accelerating MATLAB with CUDA [20]. These toolboxes obviate the need to write the above mentioned wrapper functions.

5 EVALUATION

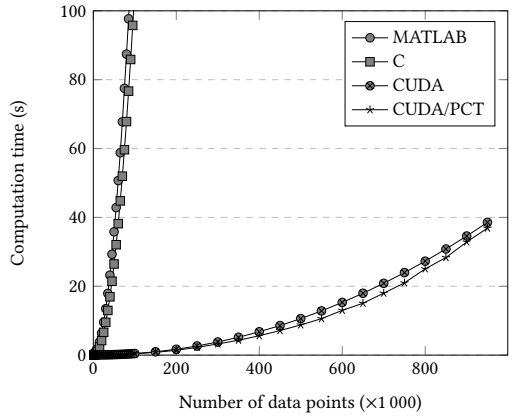
In this section we evaluate the achievable speedup with our GPU implementation. As data set, we use a one-second packet trace from a 10 Gbps link of a Tier-1 service provider [3] containing more than 400 000 packet transmissions. Time is measured in microseconds, which results in one million data points. The data set is converted into a cumulative arrival time function A. For our evaluation, we compute the deconvolution $A \oslash A$, which yields the empirical envelope of the packet trace. The evaluation was done on the Intel desktop described at the beginning of Section 4, with a Tesla C2050 computing processor.

We compare the required completion times for four different implementations:

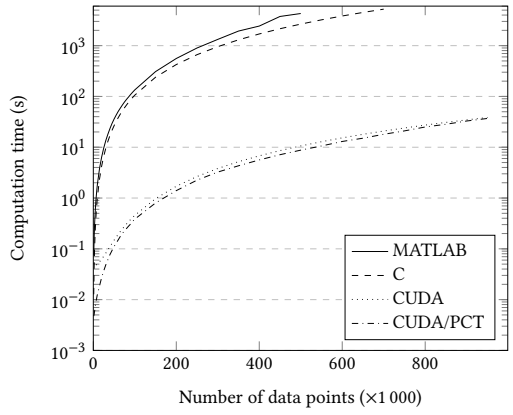
- MATLAB programming language,
- compiled C program,
- CUDA C program as described in the previous sections,
- the same CUDA C program, but invoked with the MATLAB Parallel Computing Toolbox (PCT).

The first two versions run exclusively on the CPU, and the last two version take advantage of the GPUs. All implementations are called from within MATLAB. In Figure 4 we show the computation times of all four implementations for 1 000–950 000 data points from the data trace. Since the computation times diverge quickly, we provide the results on a linear time scale (Figure 4(a)) as well as on a logarithmic time scale (Figure 4(b)).

From the figures, we observe that the computation times of methods that run on the CPU of the desktop (MATLAB and C) are



(a) Linear time scale.



(b) Logarithmic time scale.

Figure 4: Computation time of $A \oslash A$.

significantly longer than those that take advantage of the GPUs (CUDA and CUDA/PCT). The MATLAB implementation takes more than one hour to compute the empirical envelope for 500 000 data points, whereas the CUDA implementations only require around 7 seconds for the same data set. This is speedup of more than 400x. Note that our CUDA code has similar performance when invoked with the MATLAB PCT toolbox or our own wrapper function, with the customized wrapper functions resulting in a small improvement.

6 CONCLUSIONS

Applications of the network calculus must frequently perform min-plus convolutions and deconvolutions of functions. Due to their quadratic computational complexity, these operations can become impractical for large data sets. To reduce the computation time, we have developed a GPU implementation that exploits opportunities for parallel computations. We showed that, for a data set with half a million data points, our implementation exceeded a 400x speedup. We also addressed issues that arise when performing convolutions and deconvolutions of data traces, by identifying the ranges where the operations are computed correctly.

ACKNOWLEDGEMENTS

This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] S. Bondorf and J. Schmitt. The DiscoDNC v2 – a comprehensive tool for deterministic network calculus. *EAI Endorsed Transactions on Internet of Things*, 15(4), February 2015.
[2] A. Bouillard and E. Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, 2008.
[3] K. C. Claffy, D. Andersen, and P. Hick. *The CAIDA Anonymized 2011 Internet Traces – 20110825-134100.* 2011.
[4] M. Fidler, B. D. Walker, and Y. Jiang. Non-asymptotic delay bounds for multi-server systems with synchronization constraints. *CoRR*, abs/1610.06309, 2016.
[5] R. Trillo Flores and M. Boyer. Performance analysis of the disrupted static priority scheduling for AFDX. In *Proc. 12th ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 94–103, October 2014.
[6] Y. Ghiassi-Farrokhfal, C. Rosenberg, S. Keshav, and M.-B. Adjaho. Joint optimal design and operation of hybrid energy storage systems. *IEEE Journal on Selected Areas in Communications*, 34(3):639–650, 2016.
[7] T. Herpel, K.-S. Hielscher, U. Klehmet, and R. German. Stochastic and deterministic performance evaluation of automotive can communication. *Computer Networks*, 53(8):1171–1185, 2009.
[8] U. Klehmet, T. Herpel, K. S. Hielscher, and R. German. Delay bounds for can communication in automotive applications. In *Proc. 14th GI/ITG Conference - Measurement, Modelling and Evalutation of Computer and Communication Systems*, pages 1–15, March 2008.
[9] E. W. Knightly, D. E. Wrege, J. Liebeherr, and H. Zhang. Fundamental limits and tradeoffs of providing deterministic guarantees to VBR video traffic. In *Proc. ACM SIGMETRICS*, pages 98–107, May 1995.
[10] Network Research Lab. turboNC – GPU-accelerated network calculus. https://github.com/liebeherr/turbonc/, 2017. University of Toronto.
[11] K. Lampka, S. Bondorf, and J. Schmitt. Achieving efficiency without sacrificing model accuracy: Network calculus on compact domains. In *Proc. 24th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 313–318, 2016.
[12] K. Lampka, S. Bondorf, J. Schmitt, N. Guan, and W. Yi. Generalized finitary real-time calculus. In *Proc. IEEE INFOCOM*, pages 1–9, 2017.
[13] J. Y. LeBoudec and P. Thiran. *Network Calculus.* Springer Verlag, Lecture Notes in Computer Science, LNCS 2050, 2001.
[14] MATLAB. *Version 8.2 (R2013b).* The MathWorks Inc., Natick, Massachusetts, 2013.
[15] N. Moussa and R. Tourki. Using network calculus for analysing wired-wireless network on chip. In *Proc. World Congress on Computer and Information Technology (WCCIT)*, pages 1–5, June 2013.
[16] Nvidia. Nvidia CUDA C programming guide. *Nvidia Corporation*, 120(18):8, 2011.
[17] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.
[18] J. B. Schmitt and F. A. Zdarsky. The DISCO network calculator: a toolbox for worst case analysis. In *Proc. 1st International Conference on Performance Evaluation Methodolgies and Tools (VALUETOOLS)*, October 2006.
[19] K. Wang, F. Ciucu, C. Lin, and S. H. Low. A stochastic power network calculus for integrating renewable energy sources into the power grid. *IEEE Journal on Selected Areas in Communications*, 30(6):1037–1048, 2012.
[20] B. Zhang, S. Xu, F. Zhang, Y. Bi, and L. Huang. Accelerating MatLab code using GPU: A review of tools and strategies. In *Proc. 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*, pages 1875–1878, 2011.