Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Talbot Suite: a parallel software collection for the numerical inversion of Laplace Transforms

L. Antonelli, S. Corsaro, Z. Marino, M. Rizzardi

RT-ICAR-NA-2012-09 **Novembre 2012**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Talbot Suite: a parallel software collection for the numerical inversion of Laplace Transforms *

L. Antonelli [1], *S. Corsaro* [2,1], *Z. Marino* [2], *M. Rizzardi* [3]

Rapporto Tecnico N.: **RT-ICAR-NA-2012-09**

Data: **Novembre 2012**

---

* Rapporto Tecnico sottomesso a pubblicazione su rivista

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Napoli, via P. Castellino, 111, 80131 Napoli
[2] Dipartimento di Statistica e Matematica per la Ricerca Economica, Università degli studi di Napoli "Parthenope", via Medina 40, 80133 Napoli
[3] Dipartimento di Scienze Applicate, Università degli studi di Napoli "Parthenope", Centro Direzionale, Isola C4, 80143 Napoli

Talbot Suite: a parallel software collection for the numerical inversion of Laplace Transforms.

L. Antonelli, S. Corsaro, Z. Marino, M. Rizzardi

Abstract

This report presents `Talbot Suite`, a parallel software collection for the numerical inversion of Laplace Transforms, based on Talbot's method. It is designed to fit both single and multi-point Laplace inversion problems, which arise in several application and research fields.
High accuracy and efficiency are reached making full use of modern High Performance Computing (HPC) architectures. `Talbot Suite` is oriented to modern hybrid architectures. Different software versions have been implemented in the collection, specifically designed for distributed memory (MPI-based implementation), shared memory (OMP-based implementation) and hybrid (combined MPI/OMP-based implementation) systems.
In this paper we discuss our design guidelines, the software organization and we report some performance results.

1 Introduction

In this paper we discuss the development of a parallel software collection for the numerical inversion of Laplace Transforms (LT). The *Laplace Transform* $F(s)$ of a function $f(t)$ is defined as follows

$$F(s) = \mathscr{L}\left[f(t)\right] = \int_0^\infty e^{-st} f(t)\,\mathrm{d}t, \qquad \mathrm{Re}(s) > \sigma_0$$

where the *abscissa of convergence* σ_0 is defined in such a way that the integral converges uniformly if $\mathrm{Re}(s) > \sigma_0$. The Inverse Laplace problem is that of reconstructing $f(t)$ from $F(s)$.
The importance of this research topic arises from the applicability of the Laplace Transform to the solution of Differential Equations (DE), so that it covers all the areas of engineering and science whose phenomena are modelled by DE. Among them we cite fields such as electromagnetic theory, to which pioneering applications refer, wave propagation problems and finance. In particular, most applications in finance concerning *option pricing* recognized that LT approach is able to provide more accurate results than Monte Carlo simulations under certain circumstances [6].

Numerical inversion of Laplace Transforms is well-known to be an ill-conditioned problem [5, 4]. For this reason, to obtain reliable results, time-consuming processes are often required: this makes them of no practical use in some contexts, for instance in financial applications, where time-to-response is a crucial factor. High performance resources and methodologies allow to develop efficient inversion algorithms. Moreover, parallelism is now affecting all kinds of software development processes, from large-scale numerical simulations to desktop commodity applications, because of the paradigm shift towards multi-core technologies, thus raising the request of HPC software. Nevertheless, not much has been done in this framework. In [25] a parallel software for distributed memory environments for the numerical inversion of the LT is presented. The author proposes a parallel algorithm based on a Fourier series method. In [7] a parallel algorithm for Talbot's method, designed for distributed memory machines, is introduced. In [11] an implementation of another well-known LT inversion method, the Weeks method, is presented. GPU acceleration is used to speed-up the algorithm for the selection of the two free parameters of the method. A MATLAB implementation of the algorithm is also available [10]. At our knowledge, other previous works on the development of parallel software for LT inversion mainly concern with the parallelization of LT based methods for the solution of Partial Differential Equations arising from specific applications [12, 13, 20]. A parallel library for the numerical inversion of LT is actually lacking.

The work described in this paper is part of a research project aimed at producing a parallel software collection for the numerical inversion of Laplace Transforms, specifically oriented to modern hybrid architectures and designed to fit single and multi-point inversion problems, typically arising when the Laplace Transform is applied to solve a differential equation. The collection will integrate different numerical inversion methods since it is widely recognized that their effectiveness depends upon certain properties of the LT function. Our purpose is to collect methods such that a wide range of transforms can be accurately and efficiently inverted.

In this paper we describe `Talbot Suite`, a subset of the entire collection based on *Talbot's method* [23]. The development of other suites is work in progress.

2 Outline of theory and sketch of Talbot's algorithm

In this section we briefly recall some basic theoretical results on the Laplace Transform and its inverse; moreover we outline Talbot's algorithm.

Theorem 2.1 Existence of $F(s)$ *[19]*
If the function $f(t)$ is piecewise continuous for $t \geq 0$ and is of exponential order

α [1] *then its Laplace transform* $F(s) = \mathscr{L}(f)$ *exists for* $Re(s) > \alpha$ *and it converges absolutely.*

Corollary 2.2 *If* $f(t)$ *satisfies the hypotheses of Theorem 2.1, then*

$$\lim_{s\to\infty} F(s) = 0$$

Theorem 2.3 Uniqueness of $f(t)$ - *Lerch's Theorem [19]*
Suppose that the functions $f(t)$ *and* $g(t)$ *satisfy the hypothesis of Theor. 2.1 (so that both their Laplace transforms* $F(s)$ *and* $G(s)$ *exist) with a common exponential order* α*. If* $F(s) = G(s)$*, for all* $s \geq \alpha$ *then* $f(t) = g(t)$*,* $\forall t \in [0, +\infty)$*, and both* f *and* g *are continuous.*

Let us suppose that Theor. 2.3 holds, then the *Riemann Inversion Formula* gives, for $f(t)$, an integral representation in the complex plane along the *Bromwich contour*

$$B = \{s \in \mathbb{C} : \mathrm{Re}(s) = \sigma \ \wedge \ \mathrm{Im}(s) \in \mathbb{R}\}$$

expressed by

$$f(t) = \frac{1}{2\pi i}\int_B F(s)\, e^{st}\, ds = \frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty} F(s)\, e^{st}\, ds \tag{2.1}$$

where, since $F(s)$ may be considered as the analytic continuation of a complex function with singularities only in the half-plane $\mathrm{Re}(s) \leq \sigma_0$, the hypothesis $\sigma > \sigma_0$ guarantees that $F(s)$ is analytic on B. In such a way the exponential order constant is related to σ_0.

Talbot's method has been proposed by Talbot [23] and implemented in FORTRAN 77 by one of the authors [15]; it falls within the class of inversion methods based on the integration of (2.1) along a special contour.
The underlying idea is to apply the composite trapezoidal rule for approximating (2.1) where the Bromwich contour has been suitably replaced by the *Talbot contour*, which depends on the location of the singularities of $F(s)$ and on the value of the argument of $f(t)$. To do this, the following statements must hold

- the integration contour must enclose all the singularities of $F(s)$;
- $F(s)$ has to become negligible on the Talbot contour, as $s \to \infty$, so that (2.1) may be accurately approximated by the trapezoidal rule.

[1] The function $f(t)$ is said to be of *exponential order* α, as $t \to +\infty$, if there exist nonnegative constants $M > 0$, $\alpha \in \mathbb{R}$ and $t_0 \geq 0$ such that

$$|f(t)| \leq Me^{\alpha t}, \quad \text{for } t \geq t_0$$

More precisely, assumptions for *Talbot's method applicability* are:

$$\begin{array}{l} \bullet \quad \sigma_0 < \infty \\ \bullet \quad F(s) \text{ has singularities with finite imaginary parts} \\ \bullet \quad \lim\limits_{s\to\infty} |F(s)| = 0 \ \text{ uniformly in } \ \mathrm{Re}(s) \le \sigma_0 \end{array} \tag{2.2}$$

According to the second requirement in (2.2), Talbot's method cannot be applied to Laplace Transforms with infinite singularities along a vertical line.
The Talbot integration contour $L^\star_\nu$ has equation

$$L^\star_\nu \ : \ \ s = \sigma + \lambda\theta\cot\theta + i\lambda\nu\theta, \qquad \theta \in]-\pi,\, \pi[; \tag{2.3}$$

where the *geometrical parameters* λ, σ, ν, as well as the *accuracy parameter* N, that is the number of nodes in the trapezoidal rule, depend on t, on the singularities of $F(s)$, on the input accuracy requirement and on the floating-point arithmetic system. Talbot's error analysis allows to compute nearly optimal values for λ, σ, ν and N.

The approximation formula is (see [23] for details on its derivation)

$$f(t) \approx \widetilde{f(t)} = \lambda\, e^{\sigma t}\, \frac{T_N(t)}{N}$$

with

$$T_N(t) = \frac{\nu}{2} e^{\lambda t} F(\sigma+\lambda) + \sum_{j=1}^{N-1} (x_j \cos\phi_j - y_j \sin\phi_j) \tag{2.4}$$

where $\phi_j = \lambda t \nu \pi j/N$ and x_j, y_j depend on the values of F along the contour. Formula (2.4) appears as a difference between two Clenshaw sums; in the following we refer to it as *Talbot-Clenshaw sum*.
Talbot's algorithm is outlined in ALGORITHM 1, where the names `TAPAR` and `TSUM` are derived from Algorithm 682 in Collected Algorithms of ACM [1].

ALGORITHM 1: Talbot's algorithm

Input: LT function $F(s)$, t, error tolerance, singularities of $F(s)$
Output: $\widetilde{f(t)}$

1: compute the method parameters λ, σ, ν, N (`TAPAR`);
2: compute the Talbot-Clenshaw sum and $\widetilde{f(t)}$ (`TSUM`).

3 Parallelization strategy

The starting point was to analyze the performance of `TAPAR` and `TSUM` by means of TAU (Tuning and Analysis Utilities) profiler [21]. The tests were aimed at highlighting the computational cost of each step in ALGORITHM 1. Complete performance analysis of `Talbot Suite` can be found in [2, 3]; here we just report

two of them for supporting the discussion. Fig. 1 shows the percentages of the execution time spent in `TAPAR` and `TSUM`, which are called inside a driver function called itself by a main program. These times are related to the same LT function (labelled as `F24` from a data set of test functions listed in Appendix A) and the same error tolerance `tol`$= 10^{-12}$, but they refer to different values of t where $f(t)$ has to be approximated: $t = 10$ on the left side of Fig. 1 and $t = 3000$ on the right side. The former (a small value of t) requires only $N = 59$ addends in the summation, while the latter requires $N = 645087$, so that almost all the time is spent by the summation process.

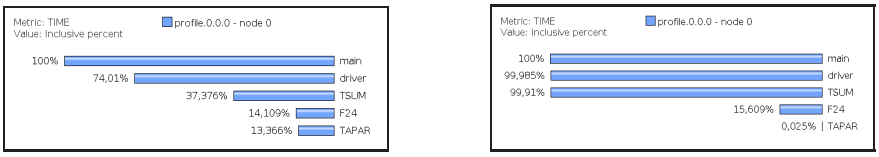

Figure 1: Percentages of the execution time in `TAPAR` and `TSUM`.

In any case, computing the sum is the most time consuming process, thus we parallelized only this step by implementing the *parallel Goertzel-Reinsch algorithm* for a Talbot-Clenshaw sum as described in [7].

Furthermore we designed `Talbot Suite` to deal with the multi-point Laplace Transform inversion for its importance in many applications. The natural strategy to parallelize a multi-point inversion problem is to distribute a set $\mathbb{T}$ of values among processes. Uniform partitioning of $\mathbb{T}$ could lead to load imbalance, since for LT with complex singularities the number of addends strongly varies with respect to $t \in \mathbb{T}$. To overcome the load imbalance problem, we implemented the *modified Talbot method* presented in [17]: this method approximates the inverse LT function $f(t)$ at several values of $t \in \mathbb{T}$ using a fixed set of parameters, estimated at an optimal $t^\star$. In the cited paper the root mean square error is proved to be minimum provided that parameters are chosen for $t^\star$ equal to the midpoint of the interval enclosing $\mathbb{T}$. In the following, we refer to Talbot's method as *classical Talbot's method* to distinguish it from the *modified* one.

Therefore we developed two parallel algorithms relying on the *classical* and *modified* methods. In such a way two levels of parallelism have been introduced: a *coarse grain parallelism* based on data partitioning (sketched in Algorithm 2) and a *fine grain parallelism* which parallelizes the summation process (sketched in Algorithm 3).
The two proposed parallel strategies may be applied in conjunction; so, aimed at hybrid architectures, we include in `Talbot Suite` a two-level parallel algorithm (sketched in Algorithm 4) based on a combination of the coarse grain and the fine grain parallelism.

4 Talbot Suite

`Talbot Suite` is written in C (double precision). We started by implementing

ALGORITHM 2: Coarse grain parallelism

Input: LT function $F(s)$, $\texttt{NTval}$, $\mathbb{T} = \{t_1, \ldots, t_{\texttt{NTval}}\}$, error tolerance, singularities of F,
midpoint $t^\star$ of the range of $\mathbb{T}$, number of processes np

Output: $\widetilde{f(t)}$, $\forall t \in \mathbb{T}$

1: compute λ, σ, ν, N for $t = t^\star$;

2: **for** *each process* i : $i = 0, 1, \ldots, np-1$ **do**

- compute $\texttt{NTval}_{\text{loc}}(i)$, the local number of t values;
- define $\mathbb{T}_i$, the local set of t values;
- **for** *each value* $t \in \mathbb{T}_i$ **do**

compute $\widetilde{f(t)}$;

end

end

ALGORITHM 3: Fine grain parallelism

Input: LT function $F(s)$, t, error tolerance, singularities of F, number of processes np

Output: $\widetilde{f(t)}$

1: compute λ, σ, ν, N;

2: compute $\widetilde{f(t)}$ in parallel.

the *classical Talbot method* included in the *Collected Algorithms of ACM* [1, 15]. The original source codes have been translated from FORTRAN 77 to C: to this purpose the code has been suitably rearranged; for example, all the `goto` statements have been removed following the structured programming paradigm and, according to the IEEE 754 (IEEE Standard for Floating-Point Arithmetic) [9], floating-point environment variables (such as *machine epsilon*, *overflow/underflow limits* and so on) have been introduced where necessary, removing all the machine constants and every call to related functions such as (FORTRAN) D1MACH [22].

`Talbot Suite` aims at HPC architectures, thus we provide implementations of parallel Talbot's algorithms (ALGORITHM 2, 3) for both shared and distributed memory environments. We employ OpenMP (OMP) [16] for the shared memory model and MPI [14] for the distributed one.
Since OMP is fully supported by several compilers (such as recent versions of `gcc`) and most PCs and Laptops are equipped with a multicore CPU, the OMP-based implementation of `Talbot Suite` runs almost everywhere with a full exploitation of computing resources. Another significant advantage is that OMP parallel codes can be executed as sequential, with no modification, on machines which do not support OMP, since the specific directives are ignored. Some preliminary performance tests motivated us to avoid, when possible, explicit

ALGORITHM 4: Two-level hybrid parallelism

Input: LT function $F(s)$, `NTval`, $\mathbb{T} = \{t_1, \ldots, t_{\texttt{NTval}}\}$, error tolerance, singularities of F,
midpoint $t^\star$ of the range of $\mathbb{T}$, number of processes np

Output: $\widetilde{f(t)}$, $\forall t \in \mathbb{T}$

1: compute λ, σ, ν, N for $t = t^\star$;

2: **for** *each process* i : $i = 0, 1, \ldots, np-1$ **do**

- compute $\texttt{NTval}_{\text{loc}}(i)$, the local number of t values;
- define $\mathbb{T}_i$, the local set of t values;
- **for** *each value* $t \in \mathbb{T}_i$ **do**

compute $\widetilde{f(t)}$ in parallel;
end

end

parallel for-loop statements. Therefore, the parallelism realized in summation functions obeys a *SPMD* programming model.

In order to take full advantage of recent hybrid architectures, we developed a parallel two-level algorithm which combines the two strategies (ALGORITHM 4) and employs MPI for the coarse grain parallelism (data partitioning) and OMP for the fine grain parallelism (parallel summation).
The organization of `Talbot Suite` is illustrated in Fig. 2.

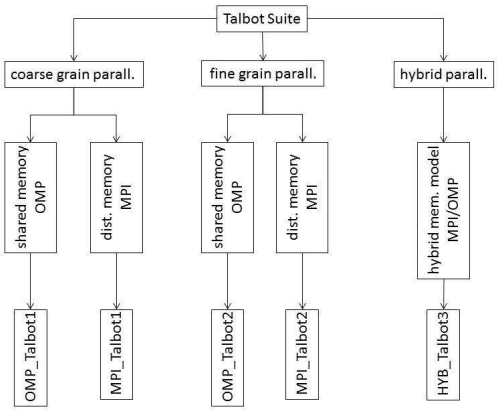

Figure 2: Structure of `Talbot Suite`.

The design of `Talbot Suite` is aimed at different end-users; we introduced a driver function at user level where all the method-specific parameters have been removed. Nevertheless, expert users can directly call skill-level functions to

customize the method according to their needs.

In order to assemble together other suites, which are work in progress, we fix some conventions (rules) as follows. Each suite refers to a particular numerical inversion method; to define a standard notation, all the functions at user level in the same suite have been named as the method with the addition of a prefix related to a specific implementation and a suffix related to a level of parallelism, while, at skill level, the names of internal functions have been modified introducing an explicit reference to the method so that they are directly related to their driver function. Moreover parameters common to all inversion methods (such as Laplace Transform function, abscissa of convergence, t, etc.) are listed first in function prototypes, followed by parameters specific to an inversion method (such as σ, λ, ν for Talbot's method) and, if needed, followed by parameters related to the parallel environment (such as the `MPI` communicator for MPI version).
The driver functions have been named as follows

`VER_Method level`

where

- `VER` is `MPI` (pure MPI), `OMP` (pure OMP) or `HYB` (hybrid MPI-OMP);
- `Method` is `Talbot` for `Talbot Suite`;
- `level` contains `1` for the coarse grain parallelism, `2` for the fine grain parallelism or `3` for the hybrid one.

For example, `OMP_Talbot2` is the driver function for the OMP-based implementation with the fine grain parallelism and `HYB_Talbot3` is the driver routine corresponding to the hybrid two-level parallel strategy.

5 Numerical results

In this section we report some results on performance of `Talbot Suite`. We perform our tests using a set of Laplace Transform functions with different analytical properties and known inverse functions [18]. Appendix A contains the complete list of LT test functions.
To summarize main results, we refer only to the following test function

$$F_{24}(s) \quad = \quad s/\left(s^2+9\right)^2 \qquad\qquad f_{24}(t) := \mathscr{L}^{-1}\left[F_{24}(s)\right] \quad = \quad t\sin(3t)/6$$

having complex polar singularities. It has been chosen because, as t becomes large, complex singularities may lead to many terms in the summation, so that $F_{24}(s)$ is suitable to test the fine grain parallelism. For moderate values of t it behaves like any other LT test function.
As regards accuracy, we just focus on the impact of parallelization on results. Because of the new recurrence formula in the parallel version of Goertzel-Reinsch algorithm, the fine grain parallelism may slightly alter the accuracy with respect

to the sequential algorithm. In Fig. 3 the relative error of a multi-point inversion problem with $t \in]0, 10000]$ is reported and the error tolerance, `tol`$= 10^{-12}$, is represented by the horizontal line. On the left side, the plot refers to the sequential classical Talbot method and, on the right side, the three plots refer to the fine grain parallel algorithm with 64, 96 and 128 processes respectively. Errors from sequential and parallel algorithms show a similar behavior.

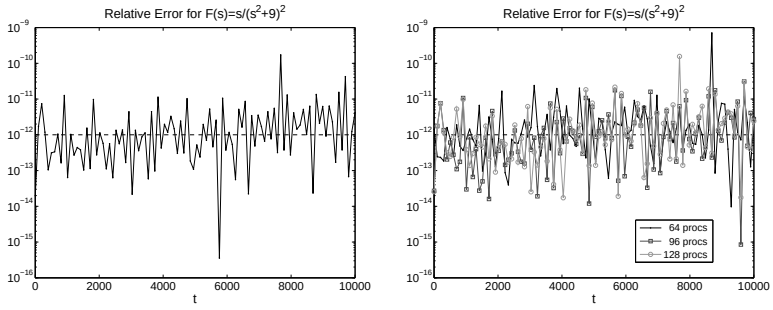

Figure 3: Tolerance and local errors in sequential and parallel *Goertzel-Reinsch* algorithm.

In order to test efficiency under different conditions, we chose for t the intervals $[10, 50]$ and $[1000, 3000]$ since, as already pointed out, the number of terms in the sum strongly increases with respect to t for LT functions with complex singularities. Moreover, two problem sizes have been applied to each interval: that of inverting a LT at a few tens or at some hundreds of points, suitable to test the coarse grain parallelism.
In Fig. 4 the number N of summands in *classical* and *modified* Talbot's methods is plotted, for $F_{24}(s)$ and `tol`$= 10^{-12}$, to emphasize its dependence on t.

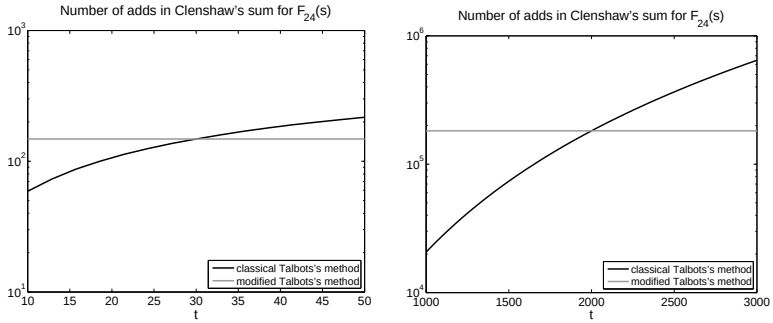

Figure 4: Number of terms in the Talbot-Clenshaw sums for classical and modified Talbot's method.

The horizontal straight line refers to the modified method, the other is related to the classical method. In the same figure the two pictures compare the above-mentioned intervals: on the left side, the interval $[10, 50]$, with moderate values

num. of t values	$t \in [10, 50]$		$t \in [1000, 3000]$	
	classical	*modified*	*classical*	*modified*
24	3473	3552	5679999	4367976
120	17385	17760	27976610	21839880

Table 1: Total number of terms in the Talbot-Clenshaw sum for `tol`$= 10^{-12}$ and $F_{24}(s)$.

of t, requires a summation with approximately a hundred of terms while, on the right side, the interval $[1000, 3000]$, with larger values, requires from 10^4 to 10^6 addends.
We recall that the modified method computes only once the parameters at the middle point of the interval, while the classical one repeats this computation for each t. However, the former may require a larger total number of additions since N does not vary linearly with t in the classical method. Tab. 1 reports the values of N used by the two methods when $F_{24}(s)$ has to be inverted for 24 or 120 values of t in the selected intervals; these test problems are discussed in the following.

We now focus on efficiency exhibited by the parallel algorithms. In order to test the impact of architectures on software performance, we ran our tests on different machines with different compilers; in this paper we refer to the following ones:

- **Blade Server**: it consists of 4 blades, each one equipped with

  CPU Intel Xeon Quad-Core E5540 2.53 GHz processor with Hyper-Threading Technology.

  RAM 6 GB and three levels of cache memory (8 MB L3 shared cache memory and 256KB L2, 32KB L1 cache memories for each core).

  Connection 1 Gigabit Ethernet network.

  Software Compiler: GNU gcc v. 4.6.1 [8]; MPI API: mpicc/mpiexec for MPICH2 version 1.4.

- **Beowulf Cluster**: it consists of 12 nodes, each one equipped with

  CPU Intel Core2 Quad-Core Q9550 2.83GHz processor [2].

  RAM 8 GB and 12 MB L2 cache shared memory.

  Connection 1 Gigabit Ethernet network.

  Software Compiler: PGI pgcc v.12.3 [24] and GNU gcc v. 4.4.3; MPI API: mpicc/mpiexec for MPICH2 version 1.4.1.

[2] This processor is not equipped with Hyper-Threading Technology.

Almost all the results reported in this paper refer to Blade Server. Only for the hybrid implementation we present results obtained on both the systems, since its performance is more dependent on the computing environment than the other implementations.

Different strategies for mapping MPI processes can be considered. We tested three of them: the first one, *by node*, also known as *round robin mapping*; the second strategy, *by slot*, assigns processes to each blade/node up to the core level; the third strategy fills up to the processing unit (PU) level. On Blade Server the first mapping strategy exhibited, in most cases, the largest efficiency, thus results given below refer to it. For the sake of brevity we do not report here the results concerning comparison among the mapping strategies, referring readers to [3].

For the OMP implementation up to eight parallel processes have been activated taking full advantage of quad-core processors equipped with Hyper-Threading Technology. Moreover static and dynamic runtime adjustment of the number of threads within a team has been tested: with GNU gcc compilers better performance has been achieved by setting it to "static", while for PGI pgcc compiler no significant difference has been observed. Even in the HYB implementation the runtime adjustment has no effect. The reported results refer to static adjustment of the number of threads.

Figs. 5 and 6 compare, for MPI and OMP implementations respectively, Algorithms 2 (coarse grain parallelism) and 3 (fine grain parallelism) when applied to test cases of Tab. 1.

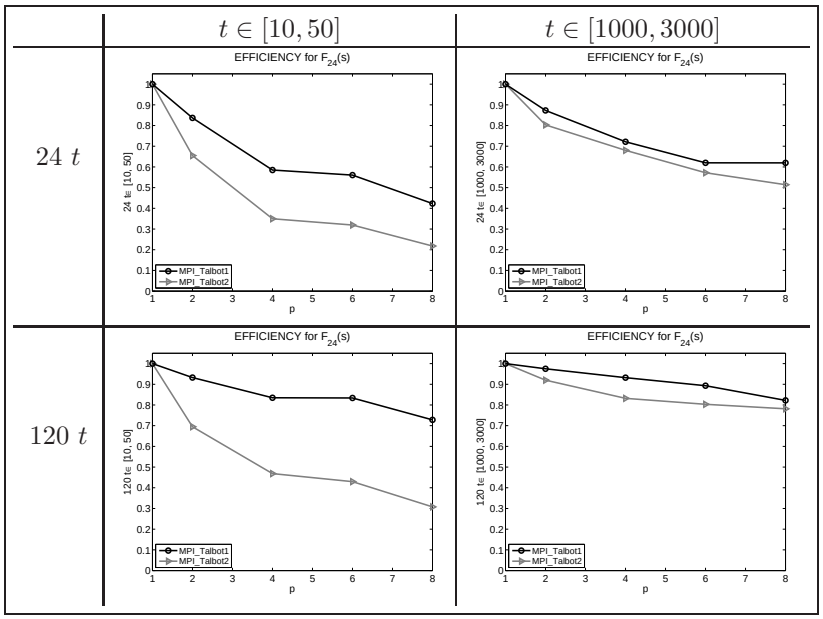

Figure 5: Efficiency of `Talbot Suites`'s MPI implementation.

In the case 24 $t \in [10, 50]$ efficiency is acceptable only when the number of processes is small since there are a few values of t and their moderate values involve Talbot-Clenshaw sums with a few addends. Figures clearly show that parallelism becomes effective when we increase just one of the two parameters or both of them.
The coarse grain parallel algorithm (`VER_Talbot1`) is particularly designed for multi-point inversion problems and, since it is embarrassingly parallel, exhibits a good efficiency especially when the number of t values is large. It usually outperforms the fine grain parallel algorithm (`VER_Talbot2`), but the gap between them decreases for larger values of N; in fact, the latter requires just a single reduction operation to collect the local Talbot-Clenshaw sums, so communication overhead decreases with respect to the number N of addends.

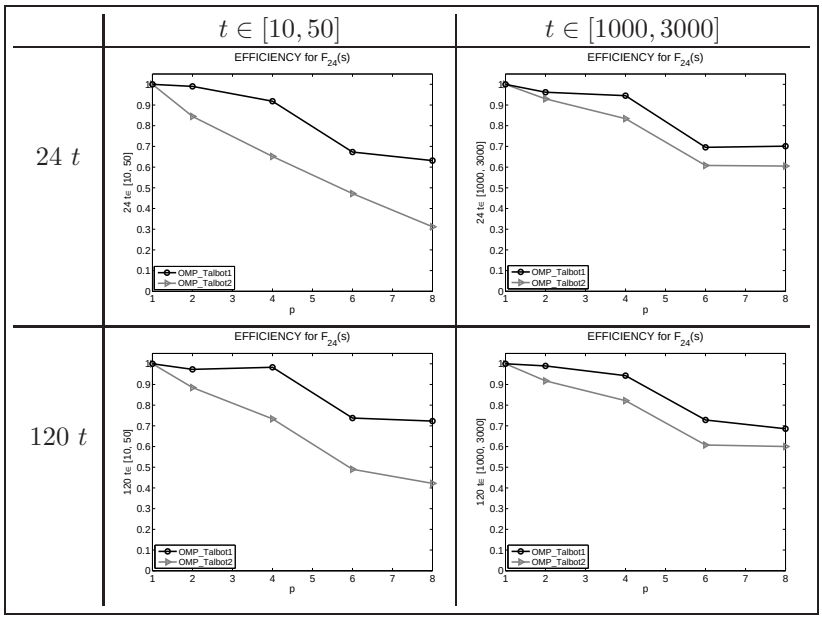

Figure 6: Efficiency of `Talbot Suites`'s OMP implementation.

On the other hand, when the numerical inversion is required for a few large values of t, the fine grain parallel algorithm allows to preserve the point-wise accuracy of classical Talbot's method with levels of efficiency comparable to the coarse grain one. In Fig. 6 performance exhibits a slight decline as soon as Hyper-Threading is activated (p$>$ 4).

Fig. 7 reports the efficiency of the HYB implementation, measured on Blade Server and on Beowulf Cluster, for the largest problem size applied to the largest interval.
Different combinations of MPI processes (coarse grain parallelism) and OMP threads (fine grain parallelism) are considered; on the horizontal axis the overall

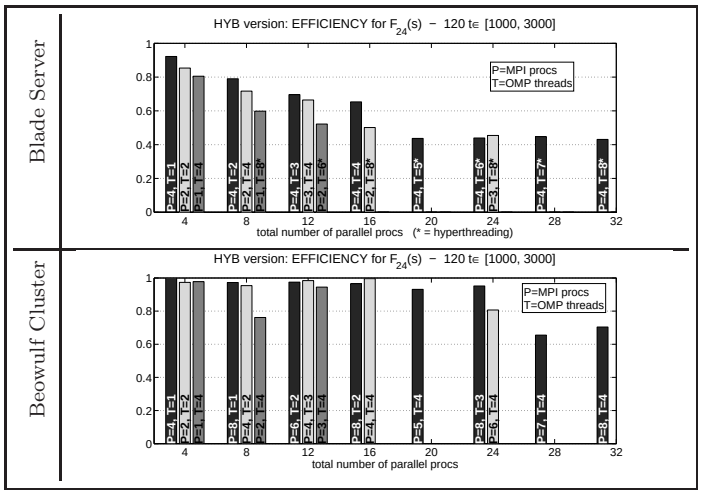

Figure 7: Efficiency of `Talbot Suite`'s HYB implementation on Blade Server with gcc and on Beowulf Cluster with pgcc.

number of parallel processes involved in the computation is reported; bar labels specify the number of MPI processes (P) and the number of OMP threads (T) related to P×T total parallel processes: for instance, the label "P=3, T=4" refers to a run with 3 MPI processes and 4 OMP threads. These results show that performance depends on the particular combination of P and T, on process mapping strategies as well as on the particular problem to be solved, i.e. a combination of the number of t values and of addends. We also emphasize that efficiency on Blade Server reduces for $T > 4$, due to Hyper-Threading.

Comparing efficiency in Fig. 7, we note that performance strongly varies on the two systems with different compilers. Since the gcc compiler is available on Beowulf Cluster too, we tested the HYB implementation also with this compiler. Using the gcc compiler, performance is similar on both the systems. So we argue that the use of the pgcc compiler motivates the better performance, observed on Beowulf Cluster with respect to Blade Server, probably because it manages for-loop statements more efficiently than gcc. Another reason for this lies in activation of Hyper-Threading, which occurs on Blade Server only. For example, although the number of parallel processes goes up to 32=P×T on both the architectures, for Beowulf Cluster it is given by P=8, T=4 (node's CPUs do not offer Hyper-Threading) while for Blade Server it is given by P=4, T=8 (blade's CPUs are equipped with Hyper-Threading).

6 Concluding remarks

In this report we present `Talbot Suite`, a C software collection for the numerical inversion of Laplace Transforms, based on classical and modified Talbot's method.

Its release is the first stage of a research project aimed at developing a parallel software collection, which will integrate different numerical inversion methods such that a wide range of transforms can be accurately and efficiently inverted. The motivation for this project is the current lack of a parallel software for this problem, in spite of its applicability to several research fields.

`Talbot Suite` has been designed to fit both single and multipoint LT inversion problems, which are typically modelled by differential equations. We developed both a fine grain and a coarse grain parallel algorithm. The former is preferable when the inverse LT has to be approximated at a few points with very large values, while the latter is preferable when the number of inversions is high.

To benefit from modern HPC architectures, `Talbot Suite` provides three different subcollections specifically designed for distributed memory, shared memory and hybrid systems: they consist of a MPI-based implementation, an OMP-based implementation and a combined MPI/OMP-based one respectively. The hybrid one implements the coarse grain parallel algorithm by means of MPI and the fine grain parallel algorithm by means of OMP.

Programming guidelines and software organization are described in the paper. Moreover, in order to share our experiences, some efficiency results, from tests carried out on different hardware/software systems, have been discussed.

References

[1] ACM CALGO. Collected Algorithms of ACM. http://calgo.acm.org/.

[2] Antonelli, L., Corsaro, S., Marino, Z., and Rizzardi, M. 2012a. Performance Profiling of Talbot Suite with TAU Analysis Tools. Part 1: OMP functions. Tech. Rep. RT-ICAR-NA-2012-4, ICAR-CNR.

[3] Antonelli, L., Corsaro, S., Marino, Z., and Rizzardi, M. 2012b. Performance Profiling of Talbot Suite with TAU Analysis Tools. Part 2: MPI functions. Tech. Rep. RT-ICAR-NA-2012-7, ICAR-CNR.

[4] Bellman, R. E. and Roth, R. S. 1984. *The Laplace Transform.* World Scientific, Singapore.

[5] Cohen, A. M. 2007. *Numerical Methods for Laplace Transform Inversion.* Springer-Verlag, New York.

[6] Craddock, M. J., Heath, D. P., and Platen, E. 2000. Numerical inversion of Laplace Transforms: a survey of techniques with applications to derivative pricing. *Journal of Computational Finance 4,* 1, 57–81.

[7] de Rosa, M. A., Giunta, G., and Rizzardi, M. 1995. Parallel Talbot's Algorithm for distributed memory machines. *Parallel Comp. 21*, 783–801.

[8] Free Software Foundation Inc. GCC, the GNU Compiler Collection. http://gcc.gnu.org/.

[9] Kahan, W. 1997. IEEE Standard 754 for Binary Floating-Point Arithmetic. Tech. rep., Elect. Eng. & Computer Science, Univ. of CA.

[10] Kano, P. 2011. Weeks' Method for Numerical Laplace Transform Inversion with GPU acceleration. MATLAB File Exchange, file id:30965.

[11] Kano, P. and Brio, M. 2011. C++/CUDA Implementation of the Weeks Method for Numerical Laplace Transform Inversion. Acunum Algorithms and Simulations, LLC.

[12] Lai, C. H., Crane, D., and Davies, A. 2007. On a Parallel Time-domain Method for the Nonlinear Black-Scholes Equation. In *Domain Decomposition Methods in Science and Engineering XVI*, T. Barth, M. Griebel, D. Keyes, R. Nieminen, D. Roose, T. Schlick, and O. Widlund, Eds. Vol. 55. 659–666.

[13] Lee, J. and Sheen, D. 2006. A Parallel Method for Backward Parabolic Problems Based on the Laplace Transformation. *SIAM J. Numer. Anal. 44*, 1466–1486.

[14] Mathematics and Computer Science Division, Argonne National Laboratory. The Message Passing Interface (MPI) standard. http://www.mcs.anl.gov/research/projects/mpi/.

[15] Murli, A. and Rizzardi, M. 1990. Talbot's method for the Laplace Inversion Problem. *ACM Trans. on Math. Soft. 16,* 2, 158–168.

[16] OpenMP Architecture Review Board. The OpenMP API specification for parallel programming. http://openmp.org/wp/openmp-specifications/.

[17] Rizzardi, M. 1995. A Modification of Talbot's Method for the Simultaneous Approximation of Several Values of the Inverse Laplace Transform. *ACM Trans. Math. Softw. 21,* 4, 347–371.

[18] Rizzardi, M. 2012. The database of Laplace Transform test functions. Tech. Rep. TR2012-02, DSA - Università degli Studi di Napoli *Parthenope*.

[19] Schiff, J. L. 1999. *The Laplace Transform: Theory and Applications.* Springer-Verlag, New York.

[20] Sheen, D., Sloan, I. H., and Thomée, V. 2003. A parallel method for time discretization of parabolic equations based on Laplace transformation and quadrature. *IMA Journal of Numerical Analysis 23,* 2, 269–299.

[21] Shende, S. S. and Malony, A. D. 2006. The TAU parallel performance system. *The International Journal of High Performance Computing Applications 20,* 2, 287–311.

[22] slatec. Download slatec/src/D1MACH.f from Netlib Repository. http://www.netlib.org/slatec/src/.

[23] Talbot, A. 1979. The Accurate Numerical Inversion of Laplace Transforms. *J. Inst. Math. Appl. 23*, 97–120.

[24] The Portland Group Inc. PGI Parallel Fortran, C and C++ Compilers. http://www.pgroup.com/.

[25] Xiaoyong, Z. 2005. Parallel FORTRAN-MPI software for numerical inversion of the Laplace Transform and its application to oscillatory water levels in groundwater environments. *Environmental Modelling & Software 20,* 3, 279–284.

A The database of test functions

Laplace Transform test functions $F_n(s)$ with real singularities

n	Laplace Transform function $F(s)$	σ_0	sings	mult	Inverse Laplace Transform function $f(t)$
01	$1/s$	0	0	1	$u(t) = \begin{cases} 0 & t<0 \\ 0.5 & t=0 \\ 1 & t>0 \end{cases}$ u unit step function
02	$1/(s+1)$	0	-1	1	e^{-t}
03	$1/(s+0.5)$	0	-0.5	1	$e^{-0.5t}$
04	$1/(s-1)$	1	1	1	e^{+t}
05	$1/s^2$	0	0	2	t
06	$\dfrac{999}{(s+1)(s+1000)} = \dfrac{1}{s+1} - \dfrac{1}{s+1000}$	0	-1 -1000	1 1	$e^{-t} - e^{-1000t}$
07	$1/(s+1)^2$	0	-1	2	$t\,e^{-t}$
08	$1/(s+1)^5$	0	-1	5	$t^4\,e^{-t}/24$
09	$1/(s-2)^5$	2	2	5	$t^4\,e^{2t}/24$
10	e^{-5s}/s	0	0	1	$u(t-5)$, u unit step function
11	$e^{-\sqrt{s}}$	0	0	0	$\dfrac{e^{-1/(4t)}}{2t\sqrt{\pi\,t}}$ $t \geq 0$
12	$e^{-4\sqrt{s}}$	0	0	0	$\dfrac{2\,e^{-4/t}}{t\sqrt{\pi\,t}}$ $t \geq 0$
13	$1/\sqrt{s}$	0	0	0	$1/\sqrt{\pi\,t}$ $t>0$
14	$\sqrt{s+\frac{1}{2}} - \sqrt{s+\frac{1}{4}}$	0	-0.5 -0.25	0 0	$\dfrac{e^{-t/4} - e^{-t/2}}{2t\sqrt{\pi\,t}}$ $t>0$
15	$\dfrac{2}{\sqrt{s}+\sqrt{s+1}} = 2(\sqrt{s+1} - \sqrt{s})$	0	-1 0	0 0	$\dfrac{1-e^{-t}}{t\sqrt{\pi\,t}}$ $t>0$
16	$\log(s)/s$	0	0	0	$-\gamma - \log(t)$ $t>0$ [Euler const $\gamma = \psi(1)$]
17	$\log\left(\dfrac{s+1}{s}\right)$	0	-1 0	0 0	$\dfrac{1-e^{-t}}{t}$ $t \geq 0$
18	$\log\left(\dfrac{s+1}{s-1}\right)$	1	-1 $+1$	0 0	$2\dfrac{\sinh t}{t}$ $t \geq 0$
19	$e^{-1/s}/\sqrt{s}$ [essential sing.]	0	0	0	$\cos\left(2\sqrt{t}\right)/\sqrt{\pi\,t}$ $t>0$

Laplace Transform test functions $F_n(s)$ with complex singularities

n	Lap. Transf. fun. $F(s)$	σ_0	sings	mult	Inv. Lap. Transf. $f(t)$
20	$1/(s^2+1)$	0	$-i$ $+i$	1 1	$\sin(t)$
21	$1/\left((s+0.2)^2+1\right)$	0	$-0.2-i$ $-0.2+i$	1 1	$\sin(t)e^{-0.2t}$
22	$1/\left(s^2+s+1\right)$	0	$0.5(-1-i\sqrt{3})$ $0.5(-1+i\sqrt{3})$	1 1	$2/\sqrt{3}\,e^{-t/2}\,\sin(t\sqrt{3}/2)$
23	$1/(s^2+1)^2$	0	$-i$ $+i$	2 2	$[\sin(t)-t\cos(t)]\,/2$
24	$s/(s^2+9)^2$	0	$-3i$ $+3i$	2 2	$t\sin(3t)/6$
25	$(s^2-1)/(s^2+1)^2$	0	$-i$ $+i$	2 2	$t\,\cos(t)$
26	$s^2/(s^3+8)$	1	-2 $1-i\sqrt{3}$ $1+i\sqrt{3}$	1 1 1	$(e^{-2t}+2e^{t}\cos(t\sqrt{3})/3$
27	$s^3/(s^4+4)$	1	$-1-i$ $-1+i$ $+1-i$ $+1+i$	1 1 1 1	$\cos(t)\,\cosh(t)$
28	$1/(s^4-1)$	1	-1 $+1$ $-i$ $+i$	1 1 1 1	$(\sinh(t)-\sin(t))/2$
29	$\arctan(1/s)$	0	$-i$ $+i$	0 0	$\sin(t)/t \qquad t\geq 0$
30	$\log\left(\dfrac{s^2+1}{s^2+4}\right)$	0	$-i$ $+i$ $-2i$ $+2i$	0 0 0 0	$2(\cos(2t)-\cos(t))/t \qquad t\geq 0$
31	$1/\sqrt{s^2+1}$	0	$-i$ $+i$	0 0	$J_0(t)$ Bessel fun of first kind

Composite Laplace Transform test functions $F_n(s)$

n	LT function $F(s)$
101	$F_{101}(s) = F_3(s)+F_5(s)+F_{21}(s)$ $= \dfrac{1}{s+0.5}+\dfrac{1}{s^2}+\dfrac{1}{1+(s+0.2)^2}$
102	$F_{102}(s) = F_{12}(s)+F_{25}(s)+F_{29}(s)$ $= e^{-4*\sqrt{s}}+\dfrac{s^2-1}{(s^2+1)^2}+\arctan(1/s)$