J Supercomput (2017) 73:4390–4406
DOI 10.1007/s11227-017-2023-9

Modelling parallel overhead from simple run-time records

Siegfried Höfinger[1,2] · Ernst Haunschmid[1]

Published online: 31 March 2017

Abstract A number of scientific applications run on current HPC systems would benefit from an approximate assessment of parallel overhead. In many instances a quick and simple method to obtain a general overview on the subject is regarded useful auxiliary information by the routine HPC user. Here we present such a method using just execution times for increasing numbers of parallel processing cores. We start out with several common scientific applications and measure the fraction of time spent in MPI communication. Forming the ratio of MPI time to overall execution time we obtain a smooth curve that can be parameterized by only two constants. We then use this two-parameter expression and extend Amdahl's theorem with a new term representing parallel overhead in general. Fitting the original data set with this extended Amdahl expression yields an estimate for the parallel overhead closely matching the MPI time determined previously.

Keywords HPC · MPI · Parallel overhead · Performance modelling · *mpiP* · *allinea/MAP*

This study is based on work supported by the VSC Research Center funded by the Austrian Federal Ministry of Science, Research and Economy (bmwfw). The computational results presented have been achieved using the Vienna Scientific Cluster (VSC).

✉ Siegfried Höfinger
siegfried.hoefinger@tuwien.ac.at; shoefing@mtu.edu

Ernst Haunschmid
ernst.haunschmid@tuwien.ac.at

[1] VSC Team, Zentraler Informatikdienst, Vienna University of Technology, Wiedner Hauptstr. 8-10, 1040 Vienna, Austria

[2] Department of Physics, Michigan Technological University, 1400 Townsend Drive, Houghton, MI 49931, USA

1 Introduction

In times of ever increasing computing power the number of critical components affecting scalability and efficiency of a particular application is also growing rapidly. Of particular importance are the increasing dynamic nature of high-performance computing (HPC) systems [1], the necessity for a scalable yet robust implementation of the target problem using modern parallelization paradigms [2], the achievement of cache-optimal performance at the single node level [3] and a straightforward way to accurately monitor and analyse the extent to which individual system/software components do condition the overall system. It is important to raise awareness for these critical issues also at the non-specialist user level, where a great number of people nowadays is making routine use of HPC resources in order to gain new insights and drive forward exciting activities.

Assessment of parallel performance and overhead has been extensively studied in the past [4–12]. Starting with the logP model [4,5], four parameters, i.e. latency, overhead, gap (reciprocal communication bandwidth) and number of processors, could be accounted for and a given algorithm be analysed, respectively. A key design goal was to find a balance between overly simplistic and overly specific models. Application to MPI [6] and several extensions respecting large messages [6,7] and contention effects [8] have been described. A more abstract framework with tuneable complexity but still practical timing requirements has been provided with PERC [9]. More recent trends in hybrid MPI/OpenMP programming were taken care of by a combination of application signature with system profiles [10]. Along similar lines application-centric performance modelling [11,13] was described based on characteristics of the application and the target computing platform with the objective of successful large-scale extrapolation. Similar predictions could also be made with the help of run-time functions within the SUIF infrastructure [12].

Recently, the cost of computation has become cheap in relation to communication [14]; thus, in order to make an algorithm scalable, the overhead due to communication must be reduced to a minimum [15]. While several powerful tools for quantifying communication overhead have been developed in the past [16–20], their routine use by the general HPC practitioner must still be considered far from standard practice. Consequently, it would be nice to have available a quick and simple method to estimate the extent of communication overhead without the need for additional interference with the software/system layer (e.g. without recompiling, switching on profiling flags, linking to additional libraries). Ideally, such a method should be easy to adopt by any HPC user interested in the subject. In the following we aim to outline the basics and practical details of exactly such an approach.

2 Basic model

We begin our investigation with the selection of a set of scientific applications frequently used on HPC platforms. They are,

- HPL [21],
- GROMACS [22],

- AMBER [23],
- VASP [24],
- QUANTUM ESPRESSO [25],
- LAMMPS [26]
- and an in-house developed quantum chemistry code [27,28].

Realistic problems are defined and computed in parallel on increasing numbers of cores using MPI [2] as the communication protocol. Only strong scaling is considered, i.e. constant problem size computed in shorter times with increasing numbers of processing elements (cores). Times to solution, t_n, are recorded as a function of numbers of involved cores, n, and results are summarized in Tables 1, 2, 3, 4, 5, 6 and 7 (columns 1, 2). In addition, the time spent in MPI calls, τ_n^{MPI}, is also recorded and included in Tables 1, 2, 3, 4, 5, 6 and 7 (column 3). Two different tools are used to measure MPI times, in particular *mpiP* [17] and *allinea/MAP* [18]. The time records obtained from both tools are largely identical as demonstrated by the example of AMBER (see Table 3). τ_n^{MPI} assessment based on *mpiP* analysis (Tables 1, 2, 3, 4) yields individual MPI timings on a per-task basis; hence, averages need to be formed for the n different tasks of a particular sample run. Because individual MPI times do vary considerably, it was also of interest to compute the variance of τ_n^{MPI} and its corresponding standard deviation, $\pm\Delta\tau_n^{MPI}$ (see Tables 1, 2, 3, 4, column 4). Given the diversity of the appli-

Table 1 HPL: Exe-Times, t_n, and MPI-Overhead, τ_n^{MPI}

n	t_n (s)	τ_n^{MPI} (s)	$\pm\Delta\tau_n^{MPI}$ (s)	$\frac{\tau_n^{MPI}}{t_n}$
*1	1092139.0	0.0	0.0	0.000
4	243000.0	14550.0	2215.3	0.060
8	115000.0	5293.8	1557.3	0.046
16	75300.0	6988.1	2260.3	0.093
32	35200.0	4111.6	1184.4	0.117
48	27500.0	6202.5	1735.2	0.226
64	19300.0	3571.2	928.4	0.185
96	13725.0	3525.9	2313.6	0.257
128	11300.0	3400.8	668.8	0.301
160	9210.0	2963.2	434.2	0.322
192	6970.0	1749.6	360.2	0.251
256	5670.0	1738.2	309.8	0.307
320	4680.0	1455.4	294.4	0.311
384	3920.0	1204.7	231.6	0.307
640	2900.0	1213.8	185.0	0.419
768	2450.0	995.3	170.0	0.406
896	2390.0	1112.4	172.0	0.465
1024	2090.0	960.0	145.2	0.459
1296	1670.0	707.5	140.3	0.424
1520	1870.3	1084.4	122.5	0.580

mpiP evaluation; * preliminary estimate

Table 2 GROMACS: Exe-Times, t_n, and MPI-Overhead, τ_n^{MPI}

n	t_n (s)	τ_n^{MPI} (s)	$\pm\Delta\tau_n^{MPI}$ (s)	$\frac{\tau_n^{MPI}}{t_n}$
1	5317.0	0.0	0.0	0.000
2	2690.0	39.5	15.3	0.015
4	1380.0	50.7	19.1	0.037
8	730.0	43.5	10.4	0.060
16	388.0	35.6	1.4	0.092
32	251.0	68.0	55.3	0.271
48	146.0	26.6	17.6	0.182
64	134.0	42.9	28.3	0.320
80	105.0	30.4	15.9	0.290
96	121.0	52.9	12.1	0.437
112	99.1	44.6	16.5	0.451
128	78.7	30.8	14.3	0.391
160	66.6	27.3	8.3	0.410
192	98.4	61.3	6.0	0.623
224	54.0	25.4	8.3	0.470
256	53.7	27.9	7.5	0.519
288	116.0	76.0	10.5	0.655
320	63.1	40.2	5.1	0.638
336	45.6	23.8	3.5	0.522
384	47.0	27.6	3.3	0.588
416	99.4	80.6	4.1	0.811
448	44.6	27.0	5.1	0.606
512	88.1	73.5	3.5	0.834
592	43.7	29.7	1.7	0.679
688	42.0	29.4	1.7	0.699

mpiP evaluation

cations and their markedly different characteristics in terms of parallel scalability, it is not obvious to identify common trends in the introduced parallel overhead. However, what appears to be a rather general signature of all applications is the rather smooth development of the quotient between parallel overhead and total run-time, τ_n^{MPI}/t_n, which is graphically illustrated in Fig. 1 (also see final column in Tables 1, 2, 3, 4, 5, 6, 7). All data sets can be approximated by the following simple expression in two adjustable parameters, b and c,

$$\frac{\tau_n^{MPI}}{t_n} = \frac{b}{c+1} - \frac{b}{c+n} \tag{1}$$

and resulting fits are also included in Fig. 1 (solid curves). While primarily an empirical relation, Eq. (1) should still satisfy the limit value conditions, $\tau_1 = 0$ and $\frac{\tau_\infty}{t_\infty} < 1$.

Table 3 AMBER: Exe-Times, t_n, and MPI-Overhead, τ_n^{MPI}

n	t_n		τ_n^{MPI}		$\pm\Delta\tau_n^{MPI}$		$\frac{\tau_n^{MPI}}{t_n}$	
	(s)		(s)		(s)			
1	4602.0	[4623.0]	0.0	[0.0]	0.0	[0.0]	0.000	[0.000]
2	2350.0	[2362.3]	40.6	[33.1]	9.0	[–]	0.017	[0.014]
4	1230.0	[1245.5]	51.8	[59.8]	23.3	[–]	0.042	[0.048]
8	633.1	[634.4]	35.5	[34.3]	17.7	[–]	0.056	[0.054]
12	484.1	[410.9]	46.4	[41.5]	16.6	[–]	0.096	[0.101]
16	383.0	[378.3]	51.1	[43.5]	9.0	[–]	0.133	[0.115]
24	296.0	[244.6]	69.9	[50.1]	31.4	[–]	0.236	[0.205]
32	280.0	[281.2]	101.6	[100.4]	28.9	[–]	0.363	[0.357]
40	207.0	[183.5]	66.1	[59.3]	22.8	[–]	0.319	[0.323]
48	233.0	[202.9]	104.7	[79.5]	30.8	[–]	0.449	[0.392]
64	200.0	[168.0]	99.0	[71.6]	28.0	[–]	0.495	[0.426]
80	167.0	[149.1]	84.2	[69.0]	21.6	[–]	0.504	[0.463]
96	139.0	[136.7]	69.9	[67.1]	23.9	[–]	0.503	[0.491]
112	142.0	[123.4]	79.4	[62.9]	21.8	[–]	0.559	[0.510]
128	125.0	[116.4]	70.5	[62.2]	20.7	[–]	0.564	[0.534]
160	116.0	[103.7]	69.4	[58.2]	17.3	[–]	0.598	[0.561]
192	113.0	[94.9]	72.4	[55.6]	17.1	[–]	0.641	[0.586]
224	112.0	[91.4]	75.6	[55.9]	16.5	[–]	0.675	[0.612]
256	127.0	[92.1]	90.9	[59.6]	14.1	[–]	0.716	[0.647]
288	132.0	[90.6]	98.2	[60.7]	12.8	[–]	0.744	[0.670]
320	128.0	[96.5]	95.2	[66.9]	10.5	[–]	0.744	[0.693]
352	146.0	[111.3]	112.7	[82.8]	7.9	[–]	0.772	[0.744]
416	131.0	[117.0]	101.9	[89.6]	7.1	[–]	0.778	[0.766]
512	124.0	[102.1]	99.3	[79.7]	6.8	[–]	0.801	[0.781]

mpiP measurement and *allinea/MAP* evaluation in [ ] for comparison

Table 4 InHouseDev: Exe-Times, t_n, and MPI-Overhead, τ_n^{MPI}

n	t_n	τ_n^{MPI}	$\pm\Delta\tau_n^{MPI}$	$\frac{\tau_n^{MPI}}{t_n}$
	(s)	(s)	(s)	
1	2124.0	0.0	0.0	0.000
2	1290.0	172.0	0.0	0.133
4	554.0	181.7	5.8	0.328
8	356.0	196.6	8.2	0.552
16	287.0	212.9	6.3	0.742
32	239.0	203.2	3.7	0.850
64	220.0	201.9	2.7	0.918
128	215.0	206.1	1.6	0.959

mpiP evaluation

Table 5 VASP: Exe-Times, t_n, and MPI-Overhead, τ_n^{MPI}

n	t_n (s)	τ_n^{MPI} (s)	$\frac{\tau_n^{MPI}}{t_n}$
1	5375.4	0.0	0.000
2	2664.9	48.0	0.018
4	1424.9	42.8	0.030
8	889.8	42.7	0.048
16	452.9	30.8	0.068
32	261.1	46.2	0.177
48	208.0	52.4	0.252
64	159.7	48.7	0.305
80	145.0	51.5	0.355
96	126.3	44.7	0.354
112	164.0	88.1	0.537
128	107.4	47.3	0.440
160	111.1	55.4	0.499
192	109.3	60.1	0.550
224	103.4	60.6	0.586
256	104.8	66.5	0.635
512	82.3	59.2	0.719
768	86.3	69.6	0.806
1024	79.2	65.0	0.821

allinea/MAP evaluation

2.1 Generalization

So far we have been very imprecise in the use of the term "parallel overhead" and frequently replaced it with "communication overhead" or τ_n^{MPI}, etc. In general, we consider every incremental time fragment emerging within a parallel algorithm *parallel overhead* if it is in excess of the serial algorithm required to solve exactly the same type of problem. Typically this will include [15],

- time to interchange data
- time to synchronize individual parallel tasks
- extra computing time due to code sections arising only in the parallel algorithm
- computing time penalties due to load balancing issues
- computing time penalties due to inhomogeneous conditions between individual components of the parallel machine [1]

Measuring parallel overhead is not a trivial matter [29–31]. A conventional view is that to a large extent it is all covered by communication overhead. In fact, if we review the above list we see that task-level recording of individual MPI times (as done here) will either explicitly or implicitly include almost all of the incurred parallel overhead. Moreover, since our primary interest is in providing an approximate estimate, we shall consider τ_n^{MPI} to be a sufficiently accurate measure of the total parallel overhead and adopt the notation τ_n for the latter throughout the remainder of this article. Estimating

Table 6 QUANTUM ESPRESSO: Exe-Times, t_n, and MPI-Overhead, τ_n^{MPI}

n	t_n (s)	τ_n^{MPI} (s)	$\frac{\tau_n^{MPI}}{t_n}$
1	5531.3	0.0	0.000
2	3085.6	216.0	0.070
4	1784.2	269.4	0.151
8	1281.7	362.7	0.283
16	793.6	388.9	0.490
32	356.3	170.0	0.477
48	294.3	165.4	0.562
64	229.2	132.7	0.579
80	203.7	123.0	0.604
96	194.5	125.1	0.643
112	186.3	126.7	0.680
128	156.7	100.8	0.643
160	155.9	110.2	0.707
192	157.1	119.1	0.758
224	153.5	120.0	0.782
256	166.8	136.4	0.818
288	151.6	123.5	0.815
320	155.8	129.5	0.831
352	154.3	129.8	0.841
416	164.8	142.2	0.863
512	161.6	141.1	0.873
768	185.8	168.5	0.907
1024	228.3	212.3	0.930

allinea/MAP evaluation

τ_n will now help to (i) raise awareness that a particular application may be significantly affected by parallel overhead, (ii) facilitate *a posteriori* assessment of various applications reporting times to solution, t_n, as a function of numbers of cores, n, (iii) identify optimal run-time conditions on a given parallel architecture.

2.2 Solving for τ_n

In the following we shall build on the model established in Eq. (1) and try to isolate a closed form for the parallel overhead, τ_n, thereof. Starting with

$$\tau_n = t_n \left(\frac{b}{c+1} - \frac{b}{c+n} \right) \tag{2}$$

we can formally decompose the time to solution,

$$t_n = t_n^{\text{Amdahl}} + \tau_n \tag{3}$$

Table 7 LAMMPS: Exe-Times, t_n, and MPI-Overhead, τ_n^{MPI}

n	t_n (s)	τ_n^{MPI} (s)	$\frac{\tau_n^{MPI}}{t_n}$
1	4501.0	0.0	0.000
2	2432.6	73.0	0.030
4	1298.9	85.7	0.066
8	702.7	97.0	0.138
16	409.7	100.8	0.246
32	290.9	128.9	0.443
48	261.8	147.9	0.565
64	246.4	157.4	0.639
80	332.9	257.0	0.772
96	294.9	226.8	0.769
112	319.0	260.3	0.816
128	300.8	247.9	0.824
160	356.2	309.2	0.868
192	360.8	319.3	0.885
224	335.5	295.2	0.880
256	308.5	276.1	0.895
288	318.7	288.1	0.904
320	330.9	303.8	0.918
352	389.0	362.9	0.933
416	362.8	338.5	0.933
512	355.6	333.6	0.938
768	396.9	378.2	0.953
1024	423.5	402.3	0.950

allinea/MAP evaluation

into an ideal time to solution, t_n^{Amdahl}, and an associated parallel overhead, τ_n. As already implied by the superscript, the initial term is given from the classic Amdahl relation [32–35],

$$t_n^{\text{Amdahl}} = f_s t_1 + \frac{(1-f_s)t_1}{n} \tag{4}$$

where t_1 denotes the single core execution time, and f_s its serial fraction that cannot be run in parallel. It follows from Eqs. (2) and (3) that we can isolate an expression for the parallel overhead, in particular,

$$\tau_n = \frac{t_n^{\text{Amdahl}} b(n-1)}{(1+c-b)n + (b+c+c^2)} \tag{5}$$

and thus again using Eq. (3) describe the total time to solution,

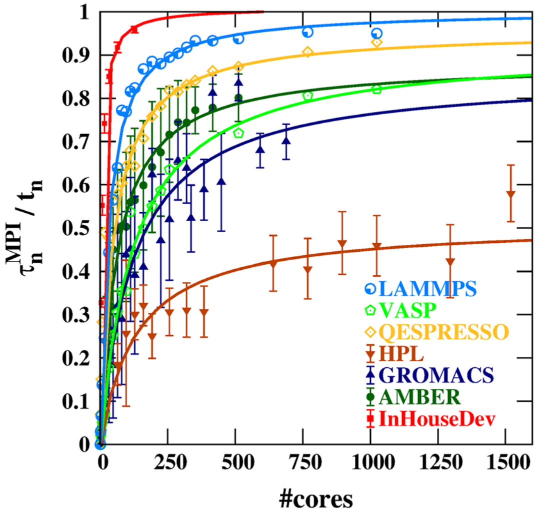

Fig. 1 Ratio of parallel overhead, i.e. the time spent in MPI communication, τ_n^{MPI}, to total execution time, t_n, for a selected set of scientific applications frequently used on HPC platforms (also see Tables 1, 2, 3, 4, 5, 6, 7). *Error bars* indicate the resulting uncertainty if we consider standard deviations to the average values of τ_n^{MPI} following *mpiP* analysis [17]. *allinea/MAP* evaluations [18] deliver mean values for τ_n^{MPI} by default. Individual data sets can be nicely fit by a two-parameter model as detailed in Eq. (1) (*solid curves*)

$$t_n = t_n^{\text{Amdahl}} \left[1 + \frac{b(n-1)}{(1+c-b)n + (b+c+c^2)} \right] \tag{6}$$

as a multiplicative extension to the original proposal of Amdahl [32–35].

3 Results

3.1 HPC systems used

All test applications examined here—except the in-house developed code—were run on the *Vienna Scientific Cluster*, version 3 (VSC-3) [36]. VSC-3 consists of 2020 compute nodes, all of them equipped with dual socket 8 core Intel Xeon CPUs (E5-2650v2, 2.6 GHz, Ivy Bridge) and interconnected by a dual-rail Infiniband QDR-80 network. Standard node memory is 64 GB; optionally available are nodes with 128 or 256 GB. The system features a rather unconventional cooling infrastructure, i.e. *Liquid Immersion Cooling* [37], where hardware components are fully immersed in mineral oil.

The in-house developed code was run on VSC-2, another HPC installation consisting of 1314 compute nodes with 2 CPUs of type AMD Opteron 6132 HE (2.2 GHz, 8 core) and again interconnected via an Infiniband QDR fabric. Standard nodes on VSC-2 provide 32 GB RAM.

3.2 Parallel overhead determined from run-time records

The simplest type of performance analysis for a particular application is to record execution times for increasing numbers of cores operating in parallel. This is also the most relevant type of analysis because it is based on exactly that type of executable that

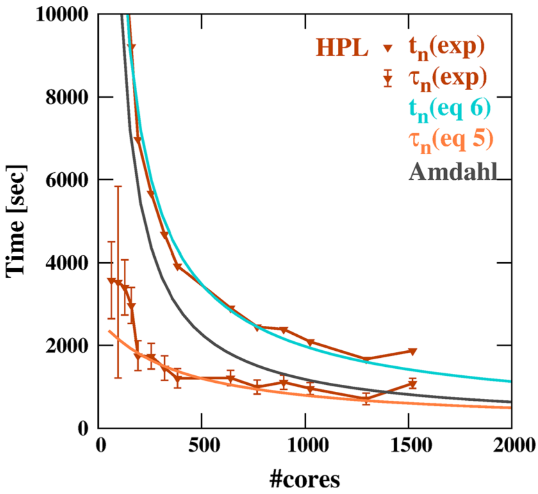

Fig. 2 Recorded times to solution, t_n, (*brown triangles*, also see Table 1, columns 1–2) as a function of numbers of cores, n, operating in parallel for application HPL [21]. Very large initial times corresponding to very small core counts have been truncated for better graphical comparison. Best fitting the data by Eq. (6) yields parameters b and c (implicitly also f_s) where the original data are reasonably well approximated (*solid line* in *cyan*). In addition, an estimate can be provided for the parallel overhead using Eq. (5) (*solid orange line*). The estimate matches the *mpiP*-derived [17] mean parallel overhead rather well (compare *orange line* to the *brown triangles* with *error bars*, respectively, Table 1, columns 3–4). Significant deviation from Amdahl's Law is seen already for small core counts (compare *cyan* to *grey line*) (Color figure online)

will be used later on in the production stage. Thus, no alterations to the binary have to be made for the purpose of analysing the code, for example introduction of internal timers, instrumentation due to profiling/debugging, inclusion of event counters, library wrappers; and all observed execution times do directly reflect the most natural run-time behaviour of the application taken into account.

Applying Eq. (6) to exactly such a simple record of just execution times, t_n, for varying numbers of cores, n, should result in the derivation of parameters, b and c, which in turn can be plugged into Eq. (5) to yield approximate estimates for the corresponding parallel overhead, τ_n. The latter is of fundamental interest, for both additional development and practical deployment at optimal run-time conditions. An example of such an approach is given in Fig. 2. The application considered was HPL [21], and the underlying data are collected in columns 1–2 of Table 1. Experimental run-times (brown triangles) are reproduced fairly well from a fit using Eq. (6). The obtained curve is shown as the cyan line in Fig. 2. Parameters b and c obtained from the fit are then applied in Eq. (5) to determine an estimate for the parallel overhead, and the corresponding graph is shown as the orange line in Fig. 2. Since in this particular case we have available experimentally derived values for τ_n (Table 1, columns 3–4), a direct comparison can be made between calculated and measured results (compare brown triangles with error bars to the orange curve in Fig. 2). Apart from an initial region of general uncertainty (see dimension of the error bars at small numbers of cores), the agreement between predicted and experimental values is rather good. A consequence

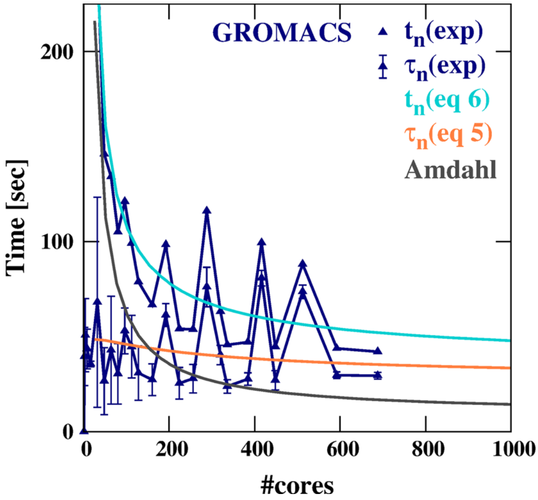

Fig. 3 Recorded times to solution, t_n, (*blue triangles*, also see Table 2, columns 1–2) as a function of numbers of cores, n, operating in parallel for application GROMACS [22]. Very large initial times corresponding to very small core counts have been truncated for better graphical comparison. Best fitting the data by Eq. (6) yields parameters b and c (implicitly also f_s) where the original data are reasonably well approximated (*solid line* in *cyan*). In addition, an estimate can be provided for the parallel overhead using Eq. (5) (*solid orange line*). The estimate matches the *mpiP*-derived [17] mean parallel overhead rather well (compare *orange line* to the *blue triangles* with *error bars*, respectively, Table 2, columns 3–4). Significant deviation from Amdahl's Law is seen already for small core counts (compare *cyan* to *grey line*) (Color figure online)

of all of this is a significant deviation from Amdahl's Law [32–35] starting already at modest numbers of cores (compare grey line with cyan curve in Fig. 2).

Additional tests were carried out for the rest of the applications, and corresponding results are graphically summarized in Figs. 3, 4, 5, 6, 7, and 8. It should be noted that the scale on both axes had to be changed considerably between different applications in order to emphasize their specific characteristics in terms of scaling and overhead times. From this it also becomes clear that the approach is rather general and can be applied to a wide range of diverse applications in identical fashion. As can be seen from Figs. 3, 4, 5, 6, 7, and 8, general results remain the same for all applications considered. However, remarkable specific differences arise upon closer examination. For example, GROMACS [22] exhibits a strange pattern of zig-zag-like execution times that is closely paralleled by the overhead times (see Fig. 3). This may indicate restricted ability to split the problem into parallel tasks of arbitrary size. Obviously, fitting such a data set can only deliver a best compromise for τ_n. In contrast, the general evolution of sample AMBER [23] appears to be smooth (Fig. 4). Similar to all the other examples, it is interesting to see how quickly τ_n is becoming the dominant factor and how steadily standard deviations to τ_n do decrease with increasing numbers of cores.

The immediate impression of the in-house developed code [27,28] is that here we certainly face the least optimized application (Fig. 5). However, it is still interesting to

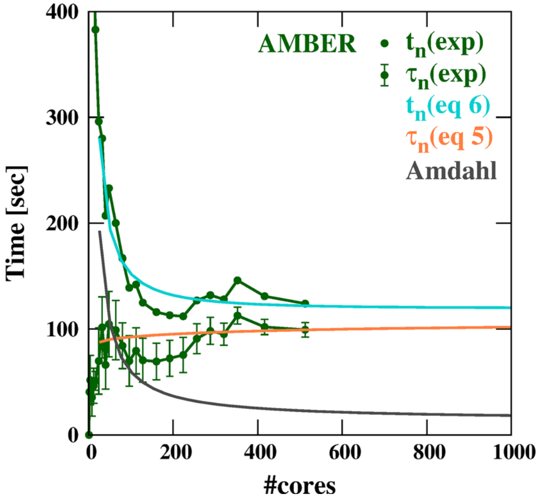

Fig. 4 Recorded times to solution, t_n, (*green dots*, also see Table 3, columns 1–2) as a function of numbers of cores, n, operating in parallel for application AMBER [23]. Very large initial times corresponding to very small core counts have been truncated for better graphical comparison. Best fitting the data by Eq. (6) yields parameters b and c (implicitly also f_s) where the original data are reasonably well approximated (*solid line* in *cyan*). In addition, an estimate can be provided for the parallel overhead using Eq. (5) (*solid orange line*). The estimate matches the *mpiP*-derived [17] mean parallel overhead rather well (compare *orange line* to the *green dots* with *error bars*, respectively, Table 3, columns 3–4). Significant deviation from Amdahl's Law is seen already for small core counts (compare *cyan* to *grey line*) (Color figure online)

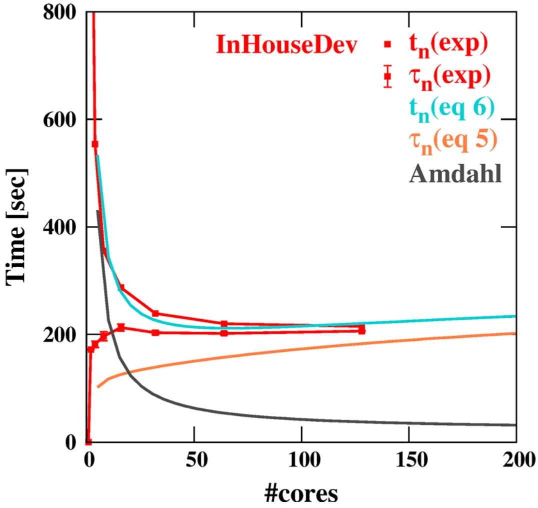

Fig. 5 Recorded times to solution, t_n, (*red squares*, also see Table 4, columns 1–2) as a function of numbers of cores, n, operating in parallel for an application developed in-house [27,28]. Very large initial times corresponding to very small core counts have been truncated for better graphical comparison. Best fitting the data by Eq. (6) yields parameters b and c (implicitly also f_s) where the original data are reasonably well approximated (*solid line* in *cyan*). In addition, an estimate can be provided for the parallel overhead using Eq. (5) (*solid orange line*). The estimate matches the *mpiP*-derived [17] mean parallel overhead fairly well for larger core counts (compare *orange line* to the *red squares* with *error bars*, respectively, Table 4, columns 3–4). Significant deviation from Amdahl's Law is seen already for small core counts (compare *cyan* to *grey line*) (Color figure online)

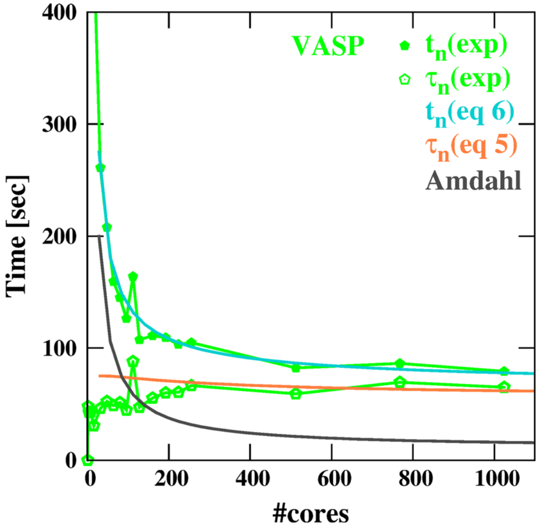

Fig. 6 Recorded times to solution, t_n, (*bright green pentagons*, also see Table 5, columns 1–2) as a function of numbers of cores, n, operating in parallel for application VASP [24]. Very large initial times corresponding to very small core counts have been truncated for better graphical comparison. Best fitting the data by Eq. (6) yields parameters b and c (implicitly also f_s) where the original data are reasonably well approximated (*solid line* in *cyan*). In addition, an estimate can be provided for the parallel overhead using Eq. (5) (*solid orange line*). The estimate matches the *allinea/MAP*-derived [18] parallel overhead fairly well (compare *orange line* to the *open pentagons* in *bright green*, respectively, Table 5, column 3). Significant deviation from Amdahl's Law is seen already for small core counts (compare *cyan* to *grey line*) (Color figure online)

observe that the proposed method for predicting parallel overhead remains applicable even in such cases. Here a saturation domain is reached quickly because of a strongly rising parallel overhead. Standard deviations of τ_n are remarkably small. Owing to the implemented communication model of primary/secondary tasks, standard deviations will start to make sense only for runs involving more than two tasks (see Table 4, fourth column). Moreover, averages and related properties will comprise only the group of secondary tasks of dimension $n - 1$.

Smooth trends are seen in sample VASP [24] with again τ_n quickly becoming the determining factor (Fig. 6). In contrast, a rather pronounced inversion in t_n evolution is observed in both of the final two samples, QUANTUM ESPRESSO [25] and LAMMPS [26] (Figs. 7, 8). Interestingly, fitted curves do still lead to reasonably good approximations of τ_n demonstrating the versatility and broad applicability of the approach.

3.3 Fitting with GNUPLOT

Care must be taken when fitting the data set and the following remarks may prove useful when reproducing our results. All our fits have been obtained with the help of package GNUPLOT [38]. Two cases may be distinguished depending on whether or not the serial fraction, f_s, is known in detail. In the majority of cases f_s in not known

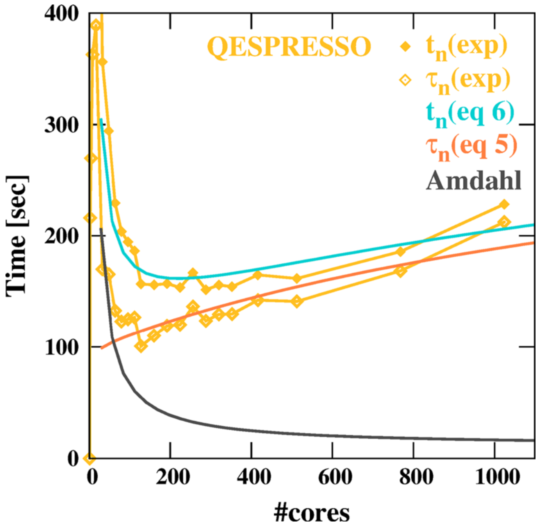

Fig. 7 Recorded times to solution, t_n, (*golden diamonds*, also see Table 6, columns 1–2) as a function of numbers of cores, n, operating in parallel for application QUANTUM ESPRESSO [25]. Very large initial times corresponding to very small core counts have been truncated for better graphical comparison. Best fitting the data by Eq. (6) yields parameters b and c (implicitly also f_s) where the original data are reasonably well approximated (*solid line* in *cyan*). In addition, an estimate can be provided for the parallel overhead using Eq. (5) (*solid orange line*). The estimate matches the *allinea/MAP*-derived [18] parallel overhead fairly well (compare *orange line* to the *open diamonds* in *gold*, respectively, Table 6, column 3). Significant deviation from Amdahl's Law is seen already for small core counts (compare *cyan* to *grey line*) (Color figure online)

and cannot be determined accurately in a quick and straightforward way. However, treating it as another entirely free parameter is also discouraged because it may rapidly turn into a negatively signed number due to over-fitting. A working procedure includes the following steps,

- define an explicit value for f_s (either known or guessed)
- fit the data using Eq. (6) and derive parameters b and c
- graphically check the quality of the fit and aim at asymptotic standard errors in the range of 10–30%
- make sure that $c > b$ and try to have $b + \Delta b \approx c$, where Δb is the reported asymptotic standard error
- incrementally decrease f_s and repeat the above steps until an optimal fit is obtained

In so doing, the formerly unknown value of f_s can be obtained as a by-product. It should be pointed out that occasionally dropping a couple of very large initial data points for small values of n was necessary to obtain a reasonable approximation in the limit of large n. Graphical control was the most important guiding principle all throughout the fitting process.

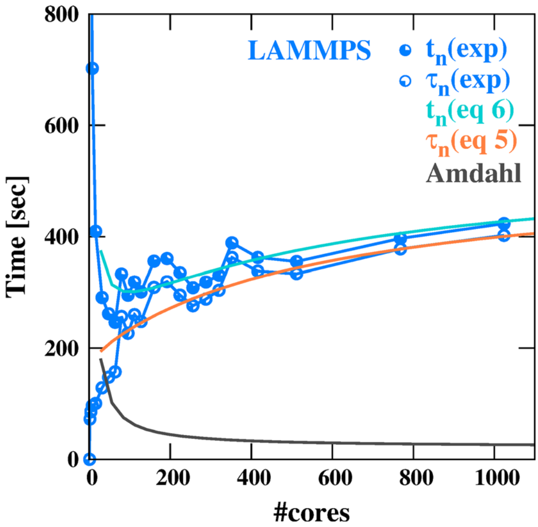

Fig. 8 Recorded times to solution, t_n, (3/4 filled discs in *bright blue*, also see Table 7, columns 1–2) as a function of numbers of cores, n, operating in parallel for application LAMMPS [26]. Very large initial times corresponding to very small core counts have been truncated for better graphical comparison. Best fitting the data by Eq. (6) yields parameters b and c (implicitly also f_s) where the original data are reasonably well approximated (*solid line* in *cyan*). In addition, an estimate can be provided for the parallel overhead using Eq. (5) (*solid orange line*). The estimate matches the *allinea/MAP*-derived [18] parallel overhead fairly well (compare orange line to the 1/4 filled discs in *bright blue*, respectively, Table 7, column 3). Significant deviation from Amdahl's Law is seen already for small core counts (compare *cyan* to *grey line*) (Color figure online)

4 Conclusion

A simple procedure is presented that allows an approximate estimation of parallel overhead solely based on run-time records. The method exhibits a broad range of applicability including well-optimized applications as well as less advanced implementations where code optimization is still in progress (compare for example Fig. 2 with Fig. 5). Asymptotic limits show a rather smooth trend and thus facilitate reasonable approximations in the limit of large n. Specifics of a particular HPC installation do not seem to play a significant role since two entirely different systems were employed here and led to similar conclusions.

Acknowledgements Open access funding provided by TU Wien (TUW).

References

1. Perarnau S, Thakur R, Iskra K, Raffenetti K, Cappello F, Gupta R, Beckman PH, Snir M, Hoffmann H, Schulz M, Rountree B (2015) Distributed monitoring and management of exascale systems in the

argo project. In: Distributed applications and interoperable systems, vol 9038. Springer, New York, pp 173–178

2. Message Passing Interface Forum MPI: a message-passing interface standard version 3.1. High Performance Computing Center Stuttgart (HLRS), Stuttgart (2015)
3. Williams S, Waterman A, Patterson D (2009) Roofline: an insightful visual performance model for multicore architectures. Commun ACM 52(4):65–76
4. Culler D, Karp R, Patterson D, Sahay A, Schauser KE, Santos E, Subramonian R, von Eicken T (1993) LogP: towards a realistic model of parallel computation. In: PPOPP ’93 Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, pp 1–12
5. Kielmann T, Bal HE, Verstoep K (2000) Fast measurement of LogP parameters for message passing platforms. In: IPDPS ’00 Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, Springer, UK, 2000, pp 1176–1183
6. Al-Tawil K, Moritz CA (2001) Performance modeling and evaluation of MPI. J Parallel Distrib Comput 61:202–223
7. Alexandrov A, Ionescu MF, Schauser KE, Scheiman C (1995) LogGP: incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation. In: Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, pp 95–105
8. Moritz CA, Frank MI (2001) LogGPC: modeling network contention in message-passing programs. IEEE Trans Parallel Distrib Syst 12(4):404–415
9. Snavely A, Carrington L, Wolter N, Labarta J, Badia R, Purkayastha A (2002) A framework for performance modeling and prediction. In: SC ’02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, IEEE
10. Adhianto L, Chapman B (2007) Performance modeling of communication and computation in hybrid MPI and OpenMP applications. Simul Model Pract Theory 15(4):481–491
11. Barker KJ, Davis K, Hoisie A, Kerbyson DJ, Lang M, Pakin S, Sancho JC (2009) Using performance modeling to design large-scale systems. Computer 42(11):42–49
12. Kühnemann M, Rauber T, Rünger G (2004) A source code analyzer for performance prediction. In: 18 International Parallel and Distributed Processing Symposium 2004, IEEE
13. Pllana S, Benkner S, Xhafa F, Barolli L (2008) Hybrid performance modeling and prediction of large-scale computing systems. In: Complex, Intelligent and Software Intensive Systems, IEEE
14. Appelhans DJ, Manteuffel T, McCormick S, Ruge J (2016) A low-communication, parallel algorithm for solving PDEs based on range decomposition. Numer Linear Algebra Appl. doi:10.1002/nla.2041
15. Grama A, Gupta A, Karypis G, Kumar V (2003) Introduction to parallel computing, 2nd edn. Addison Wesley, Reading
16. Böhme D, Hermanns MA, Wolf F (2012) In: Entwicklung und Evolution von Forschungssoftware, Rolduc, November 2011. Aachener Informatik-Berichte, Software Engineering, Shaker, pp 43–48
17. Vetter J, Chambreau C (2014) mpiP: Lightweight, Scalable MPI Profiling, version 3.4.1. http://mpip.sourceforge.net
18. MAP, release 6.0.6. Allinea Software Ltd. http://www.allinea.com/products/map (2016)
19. Tuning and Analysis Utilities. https://www.cs.uoregon.edu/research/tau/home.php (2013)
20. Paradyn Tools Project. http://www.paradyn.org (2013)
21. Petitet A, Whaley C, Dongarra J, Cleary A, Luszczek P (2012) HPL 2.1—a portable implementation of the high-performance Linpack benchmark for distributed-memory computers. http://www.netlib.org/benchmark/hpl/hpl-2.1.tar.gz
22. Abraham MJ, Murtola T, Schulz R, Páll S, Smith JC, Hess B, Lindahl E (2015) GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 12:19–25
23. Case DA, Cheatham-III TE, Darden T, Gohlke H, Luo R, Merz-Jr KM, Onufriev A, Simmerling C, Wang B, Woods RJ (2005) The Amber biomolecular simulation programs. J Comput Chem 26(16):1668–1688
24. Kresse G, Hafner J (1993) Ab initio molecular dynamics for liquid metals. Phys Rev B 47(1):558–561
25. Giannozzi P, Baroni S, Bonini N, Calandra M, Car R, Cavazzoni C, Ceresoli D, Chiarotti GL, Cococcioni M, Dabo I, Corso AD, de Gironcoli S, Fabris S, Fratesi G, Gebauer R, Gerstmann U, Gougoussis C, Kokalj A, Lazzeri M, Martin-Samos L, Marzari N, Mauri F, Mazzarello R, Paolini S, Pasquarello A, Paulatto L, Sbraccia C, Scandolo S, Sclauzero G, Seitsonen AP, Smogunov A, Umari P, Wentzcovitch RM (2009) QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. J Phys Condens Matter 21(39):395502

26. Plimpton S (1995) Fast parallel algorithms for short-range molecular dynamics. J Comput Phys 117(1):1–19
27. Höfinger S, Steinhauser O, Zinterhof P (1999) Performance analysis and derived parallelization strategy for a SCF program at the Hartree Fock Level. Lect Notes Comput Sci 1557:163–172
28. Mahajan R, Kranzlmüller D, Volkert J, Hansmann UH, Höfinger S (2008) Detecting secondary bottlenecks in parallel quantum chemistry applications using MPI. Int J Mod Phys C 19(1):1–13
29. Ezekiel J, Lüttgen G (2008) Measuring and evaluating parallel state-space exploration algorithms. Electron Notes Theor Comput Sci 198:47–61. doi:10.1016/j.entcs.2007.10.020
30. Belikov E, Loidl HW, Michaelson G, Trinder P (2012) Architecture-aware cost modelling for parallel performance portability. Lect Notes Inf P–199:105–120
31. Doerfler D, Brightwell R (2006) Measuring MPI Send and Receive Overhead and Application Availability in High Performance Network Interfaces. Lect Notes Comput Sci 4192:331–338
32. Amdahl GM (1967) Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings AFIPS '67 (Spring) Joint Computer Conference, ACM, New York, pp 483–485
33. Hill MD, Marty MR (2008) Amdahl's law in the multicore era. Computer 41(7):33–38
34. Yavits L, Morad A, Ginosar R (2014) The effect of communication and synchronization on Amdahls law in multicore systems. Parallel Comput 40(1):1–16
35. Sun XH, Chen Y (2010) Reevaluating Amdahl's law in the multicore era. J Parallel Distrib Comput 70(2):183–188
36. Vienna Scientific Cluster, generation 3. http://vsc.ac.at/systems/vsc-3
37. Green Revolution Cooling. http://www.grcooling.com
38. Bröker HB, Campbell J, Cunningham R, Denholm D, Elber G, Fearick R, Grammes C, Hart L, Heckingu L, Juhász P, Koenig T, Kotz D, Kubaitis E, Lang R, Lecomte T, Lehmann A, Mai A, Märkisch B, Merritt EA, Mikulík P, Steger C, Takeno S, Tkacik T, der Woude JV, Zandt JRV, Woo A, Zellner J (2014) Gnuplot 4.6—an interactive plotting program. http://sourceforge.net/projects/gnuplot