# Distributed multiscale computing

Borgdorff, J.

**Publication date**
2014

# 5

# Performance of distributed multiscale computing[1]

**Abstract** Multiscale simulations model phenomena across natural scales using monolithic or components-based code, running on local or distributed resources. In this work, we investigate the performance of distributed multiscale computing of components-based models, guided by six multiscale applications with different characteristics and from several disciplines. Three modes of distributed multiscale computing are identified: complementing local dependencies with large-scale resources, load distribution over multiple resources and load balancing of small- and large-scale resources. We find that the first mode has the apparent

[1]The contents of this chapter are based on:
• J. Borgdorff, M. Ben Belgacem, C. Bona-Casas, L. Fazendeiro, D. Groen, O. Hoenen, A. Mizeranschi, J. L. Suter, D. P. Coster, P. V. Coveney, W. Dubitzky, A. G. Hoekstra, P. Strand, and B. Chopard. Performance of Distributed Multiscale Simulations. *Phil. Trans. R. Soc. A*, *accepted*, 2014

benefit of increasing simulation speed and the second mode can increase simulation speed if local resources are limited. Depending on resource reservation and model coupling topology, the third mode may result in a reduction of resource consumption.

## 5.1 Introduction

Multiscale modelling and simulation is a field receiving wide interest [62], from mathematics [56], biology [41, 125, 127] physics [44, 50, 82, 118], engineering [75, 76], and many other disciplines. A small number of theoretical frameworks provide an overarching view on multiscale modelling [22, 44, 151], some of these take a scale-aware component-based modelling approach.

This work adopts one such approach, the Multiscale Modelling and Simulation Framework (MMSF) [22, 36, 67, 70] (see also Chapter 2), which defines a multiscale model as a set of coupled single scale models. The framework distinguishes between cyclic and acyclic coupling topologies, dependent on the presence or absence of feedback loops [22].

Over the last few years we have developed a large collection of multiscale models [16, 20, 53, 60, 132, 135] and have found that these multiscale models are computationally intensive. These models can be executed on a single cluster or supercomputer, however, when considering multiple coupled submodels, a single resource may not be suitable or sufficient to run all submodels. This may be because: the submodels have different (licensed) software dependencies; need specific hardware such as GPGPUs, fast I/O or a very large number of processors to compute efficiently; or need access to a local database. Even a single submodel may need more processors than are available on any one cluster. On the other hand, to simply run all submodels on an HPC resource that provides for all needs is not always possible and certainly not always efficient, since the submodels may have highly heterogeneous characteristics. In a components-based approach, submodel code may be replaced to match a given architecture without changing other parts of the model, or submodels may be distributed over the resources that fit their needs. The former approach may be desirable, but the latter is less invasive to the code and the model, and, depending on the communication overhead, would also be beneficial for efficiency.

This chapter analyses the advantages of the component-based approach and as-

sesses the overhead involved in doing distributed multiscale computing. This is motivated by the recently completed MAPPER project, which aimed to facilitate large multiscale simulations on distributed e-Infrastructure[2]. The project was driven by seven multiscale applications from the following disciplines: nano materials [132], fusion [53], biomedicine [60], hydrology [114], and systems biology [135]. In multiscale models with a cyclic coupling topology, all submodels are active during most of the simulation, whereas otherwise a scientific workflow engine is used. We divide our applications into three categories based on how they may benefit from distributed computing: (a) by increasing simulation speed by supplementing local dependencies (e.g., specific software or hardware) with large resources (e.g. supercomputers); (b) by increasing simulation speed through using more resources than available to a single computer or cluster; and (c) by increasing resource efficiency through running each submodel on appropriate computing resources. In MAPPER, we have chosen for MUSCLE 2 [23] and MPWide [59] as coupling technologies for cyclic models and the GridSpace Experiment Workbench (EW) [20, 39] for acyclic coupling topologies. These technologies all have local and distributed computing capabilities.

## 5.2 Multiscale Modelling and Simulation Framework

We define multiscale models as coupled single scale models [22, 36, 67, 70]. We characterise coupling topologies as cyclic or acyclic. A cyclic coupling topology involves feedback between single scale models, whereas acyclic coupling topologies do not. Moreover, pairs of interacting single scale models are characterised by having either temporal scale separation or overlap. According to MMSF coupling templates, submodels with temporal scale overlap exchange messages during their executions and are able to run in parallel. Indeed, they may need to run concurrently to be able to exchange data. In contrast, submodels whose time scales are separated run sequentially, so they will generally not be able to compute in parallel.

In MAPPER, we have defined a tool-chain [17, 20] to compute multiscale models that can be described with the MMSF. It starts by specifying the architecture with the Multiscale Modelling Language (MML) [22] in a dedicated user interface and then executing it with the GridSpace experiment workbench [39] for acyclic coupling topologies, and MUSCLE 2 [23] (see Chapter 3), if needed in combination with MP-

[2]MAPPER project website: `www.mapper-project.eu`

Wide [59], for cyclic coupling topologies. Distributed multiscale simulations are coordinated by middleware, in our case QCG-Broker [84] and the Application Hosting Environment [154]. Zasada *et al.* [155] describe the MAPPER infrastructure in more detail. Middleware is likely to play an important role to ease the transition to distributed computing by managing the resources from a central location and arranging co-allocated resources.

## 5.3 Performance context

When is distributed multiscale computing a valuable addition to multiscale modelling and simulation? We identify three key aspects to this question: how will the understanding and development time of a multiscale model benefit from modularisation, how long does it take to complete a simulation, and how many resources are used in the process. Ideally, the development time, time to complete a simulation (makespan), and the amount of required resources are minimised. In practice, these aspects have to be balanced, so as to not increase resources usage exorbitantly for a small gain in performance, or to sacrifice performance for the sake of lowest cost.

Already when modelling, a multiscale model may benefit from modularisation by dissecting it in multiple coupled single scale models, because this also signifies a separation of concerns common in components-based software design [9, 55]. Each submodel in a multiscale model should be independently correct, which will in some cases be easier to validate than validating an entire monolithic model at once. Better yet, a well-validated model may already exist for part of the multiscale model. Separating the multiscale model into single scale submodels also makes it easier to replace part of the model if, e.g., more detail or a faster solving method is needed. However, it may be very hard, both theoretically and computationally, to separate a model into multiple parts if these are intrinsically and closely linked. For example, two submodels that need to exchange large quantities of data every few milliseconds may benefit from faster communication methods by putting them in a single submodel and code.

Regarding the implementation of a multiscale model, having parts of the model available as separate submodels makes it possible to use techniques that are most useful for one submodel but not another, as outlined in Figure 5.1. Thus, it possible to implement a multiscale model combining several programming languages (an existing Fortran code with a C++ library) or techniques (GPU computing with scalable MPI

and OpenMP). During execution of the multiscale model, submodels should ideally run on the hardware that is best suited for them, for example, scalable MPI code on a supercomputer and a GPU code on a GPGPU cluster, and in a suitable environment, with the required software site licenses and software dependencies. All these preconditions might not be satisfied on a single machine while they may be on a (distributed) set of machines. While combining codes may help modelling and code reuse, the communication between submodels should not become a bottleneck.

Applications can be grouped based on what advantage distributed computing has for them. In the first category, *tied* multiscale simulations have at least one submodel tied to a certain machine, and by using distributed computing other submodels are no longer tied to that machine so they can run more efficiently elsewhere. In the second category, *scalable* multiscale simulations can take advantage of using more machines to run simulations faster or with a larger problem size. In the third category, *skewed* multiscale simulations may run on supercomputers but they consume fewer resources by running less demanding submodels on machines with fewer cores.

Consider a multiscale model as a set of coupled submodels $s_1, \ldots, s_n$. The time to compute a submodel depends on the architecture of the resource it runs on, and the number of cores that it uses on that resource. A submodel $s_i$ may run on architecture $a_j \in A(s_i)$, where $A(s_i)$ denotes the set of admissible architectures for $s_i$. The time to compute submodel $s_i$ on $a_j$ with $p$ cores is then $t_i(a_j, p)$. We assume that local communication time $c_{\text{local}}$ is less than distributed communication time $c_{\text{distr}}$. The makespan (total time a model takes to completion) on local resources is $T_{\text{local}}$, using $R_{\text{local}}$ CPU hours[3], the makespan on distributed resources is $T_{\text{distr}}$, using $R_{\text{distr}}$ CPU hours. The speedup $Sp$ and relative resource use $U$ of distributed computing are defined as

$$Sp = \frac{T_{\text{local}}}{T_{\text{distr}}};\ U = \frac{R_{\text{distr}}}{R_{\text{local}}}.$$

For simplicity, the performance models are reduced to submodels $s_i$ and architectures $a_i$ with $i = 1, 2$. Much more detail is possible for each of the applications individually, and this will be reported elsewhere. For our current purposes, considering two submodels on two architectures is sufficient.

[3]The number of core hours is calculated as the number of cores reserved multiplied by the time for which they are reserved. For example, on a local resource it becomes simply, $R_{\text{local}} = p \cdot T_{\text{local}}$.

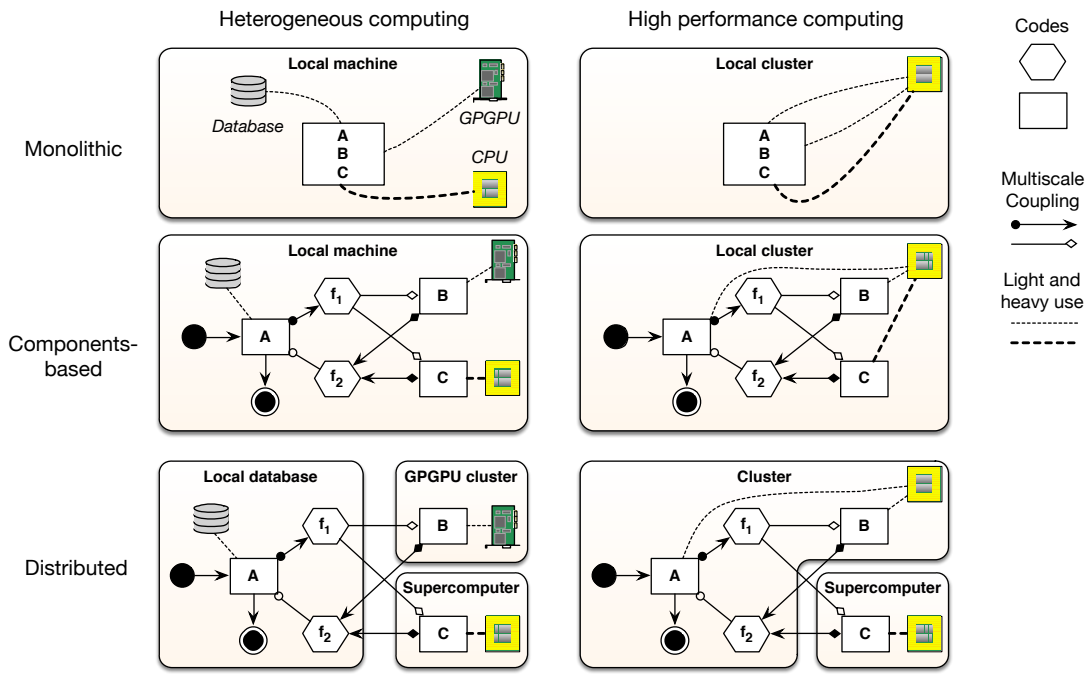


**Figure 5.1:** Scenarios to use components-based modelling or distributed computing. At the top is a monolithic model incorporating all codes *A*, *B*, *C* into one code-base. In the middle, the model is decomposed into submodels and the codes are separated by function, also separating the runtime dependencies per submodel. At the bottom shows how the components could be distributed to increase the resources effectiveness.

## 5.4 Results

The multiscale applications in this study are divided into three groups based on the benefits they derive from distributed multiscale computing, as mentioned in the introduction. The multiscale models consist of:

**tied multiscale models** a Tokamak plasma model (Transport Turbulence Equilibrium or TTE) from the fusion community [53] and a cerebrovascular blood flow model (HemeLB) from the biomedical community [60];

**scalable multiscale models** a model to reverse-engineer gene-regulatory networks (Multi-

**Table 5.1:** Resources used for performance measurements in Section 4. The total amount of cores is listed in the right-most column, although practically a fraction of that can be used in a single reservation.

| Resource | Location | Type | CPU architecture | Cores |
|---|---|---|---|---|
| Mavrino | London, UK | Cluster | Intel Xeon X3353 | 64 |
| Gordias | Geneva, Switzerland | Cluster | Intel Xeon E5530 | 224 |
| Gateway | Munich, Germany | Cluster | Intel Xeon E5-2670 | 256 |
| Scylla | Geneva, Switzerland | Cluster | Intel Xeon Westmere | 368 |
| Inula | Poznań, Poland | Cluster | AMD Opteron 6234 | 1,600+ |
| Reef | Poznań, Poland | Cluster | Intel Xeon E5530 | 2,300+ |
| Zeus | Krakow, Poland | HPC | Intel Xeon L/X/E 56XX | 12,000+ |
| Cartesius | Amsterdam, NL | HPC | Intel Xeon E5-2695 v2 | 12,500+ |
| Helios | Aomori, Japan | HPC | Intel Xeon E5-2680 | 70,000+ |
| HECToR | Edinburgh, UK | HPC | AMD Opteron Interlagos | 90,000+ |
| SuperMUC | Munich, Germany | HPC | Intel Xeon E5-2680 8C | 150,000+ |

Grain) from the systems biology community [135], and an irrigation network model (Canals) from the hydrology community [114]; and

**skewed multiscale models** a model of in-stent restenosis (ISR3D) from the biomedical community [20, 60] and a clay-polymer nanocomposites model (Nano) from the nanomaterial community [132, 133].

The details of these models can be found in Appendix C.

### 5.4.1 Tied multiscale computing

The TTE application depends on a local database and HemeLB on specific Python modules, forcing the use of low performance computing resources to execute at least part of the computations. Examples dealing with similar restrictions include the need for specific hardware or a software site license. By using distributed multiscale computing, small resources are still used to satisfy these dependencies, but they can be supplemented with larger resources where possible to decrease the simulation time.

For tied multiscale models consider the following model: $A(s_1) = \{a_1\}$, $A(s_2) = \{a_1, a_2\}$, and $t_2(a_1, p_1) > t_2(a_2, p_2)$, where $p_i$ is the number of used cores on $a_i$. Locally,

on $a_1$, the makespan would be

$$T_{\text{local,sequential}} = t_1(a_1, p_1) + t_2(a_1, p_1) + c_{\text{local}}$$

for two sequentially executing submodels and

$$T_{\text{local,concurrent}} = \max\{t_1(a_1, q), t_2(a_1, p_1 - q)\} + c_{\text{local}},$$

for two concurrently executing submodels, where $1 \leq q < p_1$ cores are assigned to one submodel, and the remaining cores to the other submodel. The resource used would be $R_{\text{local},mode} = p_1 \cdot T_{\text{local},mode}$. In a distributed setting, that would be

$$T_{\text{distr,sequential}} = t_1(a_1, p_1) + t_2(a_2, p_2) + c_{\text{distr}}, \tag{5.1}$$

$$T_{\text{distr,concurrent}} = \max\{t_1(a_1, p_1), t_2(a_2, p_2)\} + c_{\text{distr}}; \tag{5.2}$$

with $R_{\text{distr},mode} = (p_1 + p_2) \cdot T_{\text{distr},mode}$. For sequentially executing submodels, distributed multiscale computing will yield a shorter makespan if

$$c_{\text{distr}} - c_{\text{local}} < t_2(a_1, p_1) - t_2(a_2, p_2).$$

The makespan is shorter for concurrently executing submodels, if

$$c_{\text{distr}} - c_{\text{local}} < t_2(a_1, p_1 - q) - \max\{t_2(a_2, p_2), t_1(a_1, q)\}.$$

In both cases, the resource usage may increase since usually $p_1 \cdot t_2(a_1, p_1) < p_2 \cdot t_2(a_2, p_2)$, which may be acceptable if the decrease in makespan is significant.

The performance for Turbulence Transport Equation (TTE) and HemeLB is listed in Table 5.2. The TTE application needs to use a local database on the Gateway cluster (Table 5.1) from which experimental and simulation data is accessed through an application specific library. At each iteration, a short serial 1D computation is performed on such data before a 3D parallel computation is required. The database is located in a cluster in Germany with 256 available cores, but the application also has access to Helios, a community-dedicated supercomputer in Japan (Table 5.1). Per iteration, the serial part takes less than a second on the local cluster, but the parallel part takes over 390 seconds. If simulations can be distributed between Gateway and Helios, the parallel submodel can scale up to 1024 cores on such use cases, so that the

**Table 5.2:** Performance measures of tied multiscale models TTE and HemeLB. Due to the supercomputer policy restricting connections, the distributed communication speed of TTE could not be experimentally verified. Distributed communication time is estimated as $c_{\text{distr}} \approx 5$ s, based on network speeds from Germany to Japan (with a latency up to 0.5 s and throughput at least 20 MB/s).

| Simulation | $p_1$ | $T_{\text{local}}$ (s) | $p_1 + p_2$ | $T_{\text{distr}}$ (s) | Speedup | Usage |
|---|---|---|---|---|---|---|
| TTE | 128 | 397 | 16+512 | 98 | 4.0 | 1.0 |
| | | | 16+1024 | 56 | 7.1 | 1.15 |
| | 256 | 201 | 16+512 | 98 | 2.0 | 1.0 |
| | | | 16+1024 | 56 | 3.6 | 1.13 |
| HemeLB | 4 | 14481 | 4+512 | 298 | 48.6 | 2.7 |
| | | | 4+2048 | 157 | 92.2 | 5.6 |

parallel part takes less than 56 seconds, while increasing the communication time to about 9 seconds. Despite this increase, the distributed scenario is seven times as fast as the local one.

For HemeLB, a local machine with full access is used to install the necessary dependencies for part of the calculations. Since this machine only has four cores, running one iteration of a very well parallelised code there takes four hours, whereas pairing the simulation with the HECToR supercomputer reduces the runtime to a few minutes. HemeLB has been shown to scale linearly up to 32,768 cores for simulation domains of $\sim$ 90M lattice sites [61]. However, here we used a simulation domain of limited size (4.2M lattice sites). As a result, we observe an increase in resources used for the 512 core run and, especially, for the 2048 core run.

### 5.4.2 Scalable multiscale computing

The calculations of MultiGrain revolve around a multi-swarm particle swarm optimisation, which as the parameter space gets larger benefits in accuracy and convergence from larger number of particles grouped in a larger number of swarms. However, Java processes with file-based communication were used for the simulations, essentially limiting the computations to a single machine. This setup is still possible using MUSCLE 2, but if needed distributed computing can be used to involve more nodes in the computation to scale it up. For the Canals application, although the canal sections in an irrigation network are simulated with fully parallelised code, a supercomputer

or multiple clusters are necessary to simulate realistic irrigation network topologies within acceptable timespans. By using distributed multiscale computing, the total number of compute nodes may scale with the size of the network, or a single canal section may be solved faster to provide real-time feedback to a user.

Scalable multiscale models can be modelled with $A(s_1) = A(s_2) = \{a_1, a_2\}$, with $p$ cores used on both machines, and can be approached with a weak or strong scaling approach: scaling the problem size to the available resources, or keeping the problem size constant. For multiscale models in this category where $s_1$ and $s_2$ execute sequentially, there is no performance benefit, only a large increase in resource consumption. Instead we compare running $s_1$ and $s_2$ simultaneously on $a_1$ (taking time $t_1$, $t_2$), with $s_1'$ and $s_2'$ running on $a_1$ and $a_2$, respectively. Canals uses strong scaling, implying that $s_i = s_i'$, while GRNModel uses weak scaling, so that $s_i'$ does twice the number of computations as $s_i$. The modified submodels $s_i'$ take time $t_i'$.

For concurrently executing submodels, the local and distributed time are

$$T_{\text{local}} = \max\{t_1(a_1, \frac{p}{2}), t_2(a_1, \frac{p}{2})\} + c_{\text{local}} \tag{5.3}$$

$$T_{\text{distr}} = \max\{t_1'(a_1, p), t_2'(a_2, p)\} + c_{\text{distr}} \tag{5.4}$$

With weak scaling, if $t_1(a_1, \frac{p}{2}) \approx t_1'(a_1, p)$ and $t_2(a_1, \frac{p}{2}) \approx t_2'(a_2, p)$, it is possible to increase the problem size by a factor of 2 without significantly increasing the compute time, as long as the compute time is larger than the communication time. With strong scaling, if $t_1(a_1, \frac{p}{2}) > t_1'(a_1, p)$ and $t_2(a_1, \frac{p}{2}) > t_2'(a_2, p)$, and the communication time is not too long, the compute time may decrease.

The results for the applications in this category are shown in Table 5.3. For Canals, a speedup is not realised for a low resolution domain size, since the computation time is too short compared to the communication time. For a high resolution, combining the Gordias cluster with the Scylla cluster means computing the same problem 1.4 times faster, consuming 1.4 times more resources. When comparing a distributed run an equivalent monolithic model, the gain is even larger, with 1.8 times faster calculation. For time-dependent runs where high accuracy is required, distributed computing turns out to be advantageous for Canals. For MultiGrain, it simply means moving from a local desktop to the grid, by being able to use multiple nodes. With the additional computational power, it can search larger parameter spaces in a more stable timeframe, at the expense of consuming more CPU hours.

**Table 5.3:** Performance measures of scalable multiscale models Canals and MultiGrain. The Canals simulation is performed on the Gordias cluster and the Scylla cluster, with $T_{\text{local}}$ taken as the average of the $T_{\text{local}}$ of Gordias and Scylla. It is compared with running two smaller submodels (on 50+50 cores) and with running a single monolithic model (on 100 cores). The time listed for Canals is the time per iteration. The time listed for MultiGrain is the average over ten simulations and includes the standard error from the mean caused by the stochastic optimisation method used. It combines a node of the Zeus cluster and one from the Inula cluster.

| Simulation | $p_{\text{local}}$ | $T_{\text{local}}$ (s) | $p_{\text{distr}}$ | $T_{\text{distr}}$ (s) | Speedup | Usage |
|---|---|---|---|---|---|---|
| Canals (low | 50+50 | 0.015 | 100+100 | 0.023 | 0.63 | 3.2 |
| resolution) | 100 | 0.011 | 100+100 | 0.023 | 0.47 | 4.2 |
| Canals (high | 50+50 | 0.99 | 100+100 | 0.71 | 1.4 | 1.4 |
| resolution) | 100 | 1.31 | 100+100 | 0.71 | 1.8 | 1.1 |
| MultiGrain | 7 | 27±7 | 7+4 | 20±3 | 1.4 | 1.1 |
| MultiGrain | 11 | 43±16 | 11+8 | 36±10 | 1.2 | 1.5 |

### 5.4.3 Skewed multiscale computing

Although the ISR3D and Nano models run on a single large machine without problems, they do not make efficient use of the available CPUs, since some submodels scale very well while others scale hardly at all. There is a large difference between the resource usage of cyclic and acyclic coupling topologies in this case: cyclic coupling topologies involve feedback and thus force resources to be used for the duration of the entire simulation, whereas acyclic coupling topologies do not have feedback so each submodel may be scheduled for exactly the time slot that it needs. Both cluster policies and software would need to be adapted to allow online scheduling of simulations with cyclic coupling topologies, by allowing frequent short reservations, running single iterations of submodels.

The performance model is $A(s_1) = A(s_2) = \{a_1, a_2\}$, with $p_i$ resources used on $a_i$, $p_1 > p_2$, $t_1(a_1, p_1) \ll t_1(a_2, p_2)$ and $t_2(a_1, p_1) \approx t_2(a_1, p_2) \approx t_2(a_2, p_2)$. For local sequentially executing submodels, the makespan equation is

$$T_{\text{local,sequential}} = t_1(a_1, p_1) + t_2(a_1, p_2) + c_{\text{local}};$$

for concurrently executing submodels, it is

$$T_{\text{local,concurrent}} = \max\{t_1(a_1, p_1 - p_2), t_2(a_1, p_2)\} + c_{\text{local}}.$$

The resources usage becomes $R_{\text{local},mode} = p_1 \cdot T_{\text{local},mode}$.

For distributed submodels, the makespan equations become:

$$T_{\text{distr,sequential}} = t_1(a_1, p_1) + t_2(a_2, p_2) + c_{\text{distr}}, \tag{5.5}$$

$$T_{\text{distr,concurrent}} = \max\{t_1(a_1, p_1), t_2(a_2, p_2)\} + c_{\text{distr}}. \tag{5.6}$$

For both the sequential and the concurrent case there is no real benefit to makespan with distributed computing, unless submodel 2 computes much faster on another architecture ($t_2(a_2, p_2) \ll t_2(a_1, p_2)$) or if the simulation is slower due to contention between submodels when they run on the same resource ($t_1(a_1, p_1) \ll t_1(a_1, p_1 - p_2)$). The negative effects of this may be negligible if the distributed communication time ($c_{\text{distr}} - c_{\text{local}}$) is relatively small. The value may come from lower resource usage, which for the distributed case depends very much on whether the coupling topology is cyclic or acyclic:

$$R_{\text{distr},mode,\text{cyclic}} = (p_1 + p_2) T_{\text{distr},mode}, \tag{5.7}$$

$$R_{\text{distr},mode,\text{acyclic}} = p_1 \cdot t_1(a_1, p_1) + p_2 \cdot t_2(a_2, p_2) + (p_1 + p_2) c_{\text{distr}}. \tag{5.8}$$

The Nano model [133] has an acyclic coupling topology, and by running each submodel on an appropriate resource with an appropriate number of processors, its resource usage is much less than running all codes in a single reservation. This is primarily because the atomistic calculations, and especially the quantum mechanics calculations, do not run as efficiently on high core counts as the coarse-grained molecular dynamics calculations. In Table 5.4, Nano has a speedup of 1.7 (equates to multiple days) by going from a single 128 core reservation to combining that with a reservation with 1024 cores. Using multiple distributed reservations instead of one reservation of 1024 or 2048 cores, reduces the amount of resources used by 5 or 9 times, respectively.

The two most demanding submodels of ISR3D run sequentially, in a cyclic topology. Thus, simulations would not become more efficient by using distributed computing, were it not for a technique that allows the submodels run concurrently: running two simulations at once, coordinated so that their submodels alternate their execution. This increases the makespan (originally $T_{\text{local,sequential}}$) and may decrease the resource usage (originally $R_{\text{local,sequential}}$), since two simulations are calculated at once. In equa-

tions:

$$T_{\text{local,alternating}} = 2 \cdot T_{\text{local,concurrent}} > T_{\text{local,sequential}}, \tag{5.9}$$

$$R_{\text{local,alternating}} = \frac{p_1}{2} \cdot T_{\text{local,alternating}} = R_{\text{local,concurrent}}, \tag{5.10}$$

$$T_{\text{distr,alternating}} = 2 \cdot T_{\text{distr,concurrent}} > T_{\text{distr,sequential}}, \tag{5.11}$$

$$R_{\text{distr,alternating,cyclic}} = \frac{p_1 + p_2}{2} \cdot T_{\text{distr,alternating}} = R_{\text{distr,concurrent,cyclic}}. \tag{5.12}$$

The speedup stays close to 1, $Sp = \frac{T_{\text{local,sequential}}}{T_{\text{distr,alternating}}} > \frac{1}{1+\epsilon}$ for a small $\epsilon$, and the resource usage decreases, $U = \frac{R_{\text{distr,alternating,cyclic}}}{R_{\text{local,sequential}}} < 1$, if

$$|t_1(a_1, p_1) - t_2(a_2, p_2)| + 2c_{\text{distr}} - c_{\text{local}} < \epsilon \cdot T_{\text{local,sequential}}, \tag{5.13}$$

$$|t_1(a_1, p_1) - t_2(a_2, p_2)| + 2c_{\text{distr}} - c_{\text{local}} < (1 - \frac{p_2}{p_1}) T_{\text{distr,concurrent}}, \tag{5.14}$$

respectively. In words, the increase in makespan is limited and the resource usage is decreased as long as the two submodels take a similar amount of time and the distributed communication time is relatively small.

The benefit in this case is more subtle, and presents itself only on certain architectures. As shown in Table 5.4, there was a benefit for ISR3D when distributed a simulation over Huygens and Zeus [20, 60], but not when using Cartesius and Reef (see Table 5.1 for resource details). This was caused by changes in the submodel codes, making them compute one iteration faster and more efficiently, and in the hardware architectures, where a Fortran code would be slower on Huygens than on Zeus due to the compiler and processor type.

### 5.4.4 Common benefits

Besides the performance benefits outlined in the previous sections, the applications each benefit from the modularity of MML and the scale separation map [22, 67, 70]. This is clearly seen for MultiGrain, Canals, ISR3D and TTE, which make active use of the plug-and-play character of MML. The first two do this by changing the coupling topology based on the problem under study, ISR3D and TTE by easily turning on and off certain submodels for validation purposes and by interchanging similar

**Table 5.4:** Performance measures of skewed multiscale models Nano and ISR3D. The time listed for ISR3D is the time per iteration. The last two rows, for ISR3D*, concern the measurements made in Chapter 4

| Simulation | $p_{\text{local}}$ | $T_{\text{local}}$ (s) | $p_{\text{distr}}$ | $T_{\text{distr}}$ (s) | Speedup | Usage |
|---|---|---|---|---|---|---|
| Nano | 128 | $9.8 \times 10^5$ | 128+1024 | $5.7 \times 10^5$ | 1.73 | 0.88 |
| Nano | 1024 | $5.7 \times 10^5$ | 128+1024 | $5.7 \times 10^5$ | 1.0 | 0.19 |
| Nano | 2048 | $5.4 \times 10^5$ | 128+2048 | $5.4 \times 10^5$ | 1.0 | 0.11 |
| ISR3D | 144 | 281 | 144+8 | 283 | 0.99 | 1.06 |
| ISR3D vs. alt. | | | 144+8 | 531 | 0.53 | 1.00 |
| ISR3D* | 32 | 1813 | 32+4 | 1532 | 1.18 | 0.95 |
| ISR3D* vs. alt. | | | 32+4 | 1804 | 1.00 | 0.56 |

solvers with different numerical properties. For TTE, it is a way to allow combining legacy code into a modern application, whereas HemeLB is able to combine separately developed codes.

## 5.5 Conclusions

The overheads incurred by distributed multiscale computing have been discussed in the literature [16, 60]. In this study, we highlight the benefits, which clearly depend on the details of the application. We identified three types of benefits: supplementing local dependencies with HPC resources, increasing the total number of available processors, and increasing the efficiency of resource use. For tied multiscale models, the speedup is highly dependent on the power of the local resources: if the core count is high locally, the speedup will be less than if the local core count is very low, but there will be a speedup nonetheless. For scalable multiscale models, distributed multiscale computing decreases the computation time while consuming a few more resources if ratio of computation versus communication is high enough. In practice, this turns out to be at least a second of computation for every message sent. For skewed multiscale models, the main advantage of distributed computing is realised in acyclic coupling topologies, where each submodel can easily be distributed with workflow software, choosing appropriate computing resources for each step of the simulation.

However, for cyclic coupling topologies an advantage is realised only if part of a model computes faster on one resource and the other part on another. It may still

benefit from faster compute times by using more (distributed) resources, though, like in category 2. Getting more efficient simulations for cyclic coupling topologies would require a change in the way jobs are scheduled and coupling software is implemented. First of all, advance reservation would have to be used to separately schedule each iteration of a model, possibly using a task graph representation of the execution [22]. Second, a runtime environment would have to start and restart submodels for single iterations, preferably interacting with the model to get the timings of the reservations right. While the second can be implemented in software, the first also requires a policy change for existing clusters and supercomputers. The gain of this approach is that only the resources that are really needed are reserved. Since a separate reservation needs to be made for each iteration, those reservations may as well be made on several, and suitable, resources.

## Acknowledgements

The authors would like to thank Mariusz Mamoński from the Poznań Supercomputing and Networking Center, Poznań, Poland and Katerzyna Rycerz from AGH University of Science and Technology, Krakow, Poland for providing support in using the MAPPER infrastructure. We would like to thank James Hetherington en Rupert Nash from University College London, UK for their contributions to the HemeLB application.