Willem *et al. BMC Bioinformatics* (2015) 16:183
DOI 10.1186/s12859-015-0612-2

BMC Bioinformatics

METHODOLOGY ARTICLE Open Access

Optimizing agent-based transmission models for infectious diseases

Lander Willem[1,2,3*], Sean Stijven[2,4], Engelbert Tijskens[5], Philippe Beutels[1,6], Niel Hens[1,3] and Jan Broeckhove[2]

Abstract

Background: Infectious disease modeling and computational power have evolved such that large-scale agent-based models (ABMs) have become feasible. However, the increasing hardware complexity requires adapted software designs to achieve the full potential of current high-performance workstations.

Results: We have found large performance differences with a discrete-time ABM for close-contact disease transmission due to data locality. Sorting the population according to the social contact clusters reduced simulation time by a factor of two. Data locality and model performance can also be improved by storing person attributes separately instead of using person objects. Next, decreasing the number of operations by sorting people by health status before processing disease transmission has also a large impact on model performance. Depending of the clinical attack rate, target population and computer hardware, the introduction of the sort phase decreased the run time from 26 % up to more than 70 %. We have investigated the application of parallel programming techniques and found that the speedup is significant but it drops quickly with the number of cores. We observed that the effect of scheduling and workload chunk size is model specific and can make a large difference.

Conclusions: Investment in performance optimization of ABM simulator code can lead to significant run time reductions. The key steps are straightforward: the data structure for the population and sorting people on health status before effecting disease propagation. We believe these conclusions to be valid for a wide range of infectious disease ABMs. We recommend that future studies evaluate the impact of data management, algorithmic procedures and parallelization on model performance.

Keywords: Mathematical epidemiology, Agent-based model, Optimization, Performance

Background

Agent-based models (ABMs) offer endless possibilities to explore heterogeneous problems and spatial patterns but come with a large computational burden. ABMs are increasingly used to model infectious disease transmission, but little attention is given in the literature to model implementation and performance, e.g., in [1–10]. Usually the simulation time on large clusters is mentioned, but it is not clear whether computational resources are optimally used. However, computational performance is a significant aspect of a simulators' usefulness. Especially model exploration and sensitivity analysis, which require bulk calculations, benefit from efficient algorithms [11, 12]. Furthermore, improving model performance facilitates model development and testing on workstation systems.

Performance is implementation specific and therefore we compared different close-contact infectious disease simulators starting from two published ABMs for pandemic influenza: FluTE from Chao *et al.* [6] and FRED (a Framework for Reconstructing Epidemic Dynamics) from Grefenstette *et al.* [10]. Both simulators are written in C++ and are free, open source software (FOSS) under the GNU General Public License and the BSD 3-Clause, respectively. The FluTE population model consists of census tracts with communities of 2000 residents on average. The simulation runs in discrete time steps of 12-h representing daytime with work, school and day community contacts and nighttime with household and home community contacts. All children go to school in the home community

*Correspondence: lander.willem@uantwerp.be
[1]Centre for Health Economics Research & Modeling of Infectious Diseases, Vaccine and Infectious Disease Institute, University of Antwerp, Antwerp, Belgium
[2]Modeling of Systems And Internet Communication, Department of Mathematics and Computer Science, University of Antwerp, Antwerp, Belgium
Full list of author information is available at the end of the article

and adults are assigned to workplaces based on employment rates and commuting data. The community is the central unit in FluTE and one person is assigned to only one community per time step. The implementation of FRED is based on specific places for social contacts. Different places are used ranging from small households and classrooms to large schools and communities. All members of one place can have social contacts and one person might be assigned to multiple places per time step.

Individual behavior, social contact structures and population setup are very important to simulate infectious diseases. ABMs are suited to model these features because each person can be represented and stored separately. Inherent to these models are many checks and data transfers compared to the number of floating point calculations. For many years, hardware developers have been able to increase the central processing unit (CPU) performance [13]. Mass storage and memory subsystems have improved more slowly for cost reasons, which has introduced a performance gap between processing and accessing data. To reduce this imbalance, a hierarchy of small high-speed cache memories has been added to the CPU. Instead of fetching data multiple times from the main memory, it is loaded into cache and re-used [14]. The processor loads data into the cache in chunks called cache lines, which leads to efficient processing if in addition to one memory location also the nearby locations are referenced in the near future. This memory characteristic is important for the data layout of software [15]. For example, if person data is stored jointly in a person object ("Array of Structs") and next to a person's age also their gender and zip-code are checked, it will already be available in the high-speed cache. On the other hand, if person attributes are stored in separate containers ("Struct of Arrays") and only the ages are checked, many more ages are available in one cache line and less slow memory accesses are required.

High-speed memory and other advances in CPU technology have enabled performance improvements for sequential software with about a factor of two for every eighteen months during a few decades [14]. Unfortunately, these improvements have now encountered physical limits and processor manufacturers have turned to multi-core and hyper-threading architectures to increase the accumulated peak performance [16]. These novel architectures require adaptations of existing software and new programming approaches to fully exploit the performance potential. Extra attention is needed for shared resources [17] like population data or random numbers.

Random numbers are a key resource of Monte Carlo methods and the more randomness they exhibit, the better [18]. Computer algorithms are by definition deterministic procedures. They can only approximate randomness by generating a stream of so-called pseudo-random numbers. The only true randomness in a sequence of pseudo-random numbers is the "seed" value that gets the series started. The complexity increases even more with parallel simulations. Some good pseudo-random number generators (PRNG) lose their efficiency or quality, or even both, when they are parallelized [19]. In parallel applications, independent streams of random numbers are required for each thread to prevent latency. Different parallelization techniques are used in practice. In "random seeding", all processes use the same PRNG but with a different seed with the hope that they will generate non-overlapping series. More robust and versatile is the "leapfrog" method where one PRNG sequence is distributed (see Methods).

In this paper, we focus on single- and multi-core performance of discrete-time ABM simulators implemented in C/C++ to simulate infectious disease transmission. We used a limited close-contact disease simulator as case study. However, the features that we look into are also applicable to more extensive models or other types of ABMs. We investigate data management, algorithmic procedures and parallelization. We illustrate good-practice of a PRNG in a parallel context. The goal of this paper is to formulate recommendations for ABM simulators that are straightforward to realize and significantly benefit the performance.

The paper is structured as follows: First, we describe the methods starting with three different implementations of the population based on a general data structure. Second, we define an extension by adding a sorting algorithm. Third, we specify methods to run simulations in parallel with a shared-memory approach. Fourth, we describe the input data, run parameters and the work environment we used. Next, the Results and discussion section presents all findings. Finally, we end with concluding remarks and avenues for further research.

Methods

Model structure and implementation

We have opted for a model structure consisting of households, schools, workplaces and districts similar to published studies [6, 10]. Figure 1 shows a schematic overview of the locations, which represent a group of people we define as a "cluster". Social contacts can only be made within a cluster. During nighttime, people can have social contacts with members of their household and home district. During daytime, people stay at home or go to a workplace or school depending on their age, which also determines their day district. Contact between infectious and susceptible people may lead to disease transmission, which is a stochastic process based on social contact rates, infectiousness and susceptibility.

Figure 2 presents the model implementation with a general class diagram. We use a *Simulator* to organize the

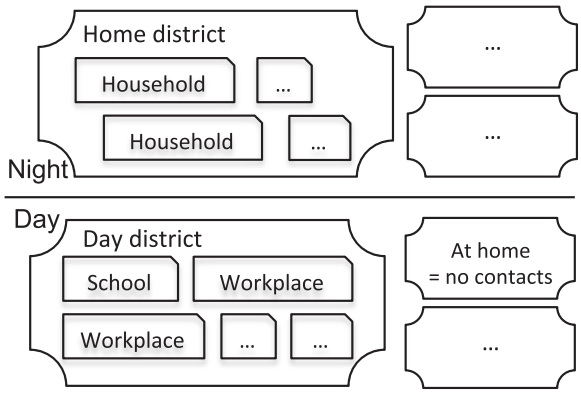

Fig. 1 Social contact structure. People are member of a household cluster and the corresponding home district at night. During daytime, people can stay at home or go to a school or workplace in a day district

activities from the people in the *Area*. The *Area* has a *Population*, different *Cluster* objects and a *Contact Handler*. The *Contact Handler* performs Bernoulli trials based on the age of the contacts and random numbers. We included a 2×2 social contact matrix, based on literature [20–22], in which the transmission rate is doubled for contacts between children (<18 y). Each *Cluster* contains links to its members. The *Population* stores all person data (id, age, household, home district, day cluster, day district and health related parameters) within or without *Person* objects but we elaborate further on this issue in the next paragraph. An infection is assumed to follow a temporal pattern of Susceptible-Exposed-Infectious-Recovered (SEIR) states similar to an influenza-like disease [6, 10]. After infection, people need 2 days of latency (infected but not infectious) before becoming infectious and 6 days to recover and acquire immunity against future infections.

We have constructed three implementations for the previous described transmission model: "FLUTE" and "FRED" are based on the corresponding open source models and "SID" has a novel data layout. The *Area* in FLUTE contains only home and day district *Clusters*. Membership to smaller sub-clusters like households, schools and workplaces can be retrieved from stored cluster IDs in *Person*. People in a district that are also member of the same sub-cluster have two opportunities for social contact and transmission. Therefore, during the processing of social contacts in the district, sub-cluster IDs need to be checked. If two people from a district are also member of the same sub-cluster, we used an aggregated transmission probability instead of performing two random draws. The *Area* of FRED and SID has also separate households and day *Clusters* (= workplaces and schools). We illustrate the difference with the following pseudo-code for the social contacts during nighttime with age dependent transmission probability P_{tr} and P_{tr*} for one or two social contacts respectively:

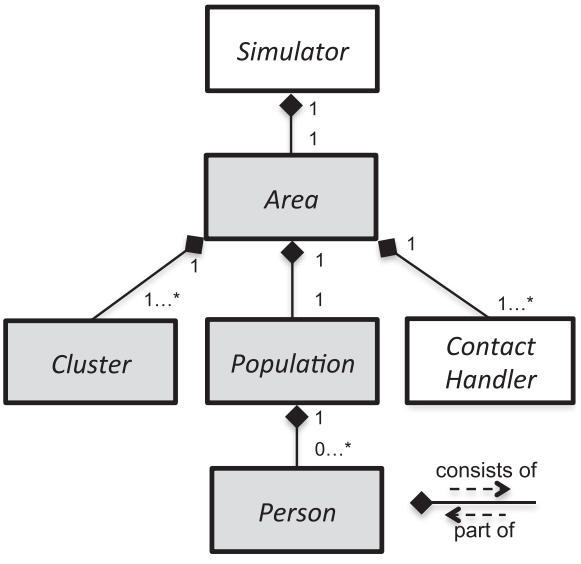

Fig. 2 Model design: classes and compositions. The digits represent the number of links that are possible. E.g., the *Area* can have 1 or many (*) *Cluster* objects, but a *Cluster* can only be part of 1 *Area*. The models differ in the implementation of the grey classes: FLUTE has less *Cluster* types in *Area* and the *Population* in SID does not contain *Person* objects

Transmission algorithm for households and home districts in FRED and SID:

1. **Loop** over all members $\{x\}$
2. **If** x is infectious
3. **Loop** over all members $\{y\}$
4. **If** y is susceptible
5. **If** random number $< P_{tr}(age_x, age_y)$
6. **Start** disease in y

Transmission algorithm for home districts in FLUTE:

1. **Loop** over all members $\{x\}$
2. **If** x is infectious
3. **Loop** over all members $\{y\}$
4. **If** y is susceptible
5. **If** x and y have equal household ID
6. **If** random number $< P_{tr*}(age_x, age_y)$
7. **Start** disease in y
8. **Else**
9. **If** random number $< P_{tr}(age_x, age_y)$
10. **Start** disease in y

Population structure

Data for an individual is stored as a *Person* object in FLUTE and FRED and the *Population* is a container of *Person* entities, stored consecutively in memory. In SID, the *Population* has a different container for each person attribute and the data of one person is always located at the same index in each of those different containers. For example, to access the age of person i in FLUTE or FRED we use "`population[i].age`" while in SID we use "`population.ages[i]`".

Population data have been extracted from the 2010 U.S. Synthetic Population Database (Version 1) from RTI International [23, 24] for Brooklyn and Nassau County, New York. Every county or state from this database can be used to obtain individual age, household, school and workplace data. People of 16 to 18 years of age with a school and work ID in the original database were assigned to the school to guarantee that people were assigned to only one day cluster. To compare different model implementations, we needed an extra social contact layer (Fig. 1). We have created home districts by adding households, sorted on ID, until a number of 2000 people was reached. We assumed that household IDs are based on geographic proximity and the threshold was adopted from Chao *et al.* [6]. The day districts have been created analogously. The Nassau population consists of 1.31 million people in 448 519 households and 140 861 day clusters. Brooklyn has 2.46 million people and the cluster sizes range from one up to 62 962 people. More details on the study populations are listed in Table 1.

The population data file determined the initial ordering of the person data in the *Population* object. We used seven different orderings for the same population details: the original sequence from the RTI database, a fully randomized order and population data sorted according to household, day cluster, and both household (first) and day cluster (second), and vice versa. To minimize the effect of random draws, we created five different files for each ordering with a random component.

Table 1 Population statistics. Legend: [min - max] and (median)

Name	Nassau, New York	Brooklyn, New York
Ages	[0 – 94] years	[0 - 94] years
Day districts	386	630
Home districts	656	1231
Day clusters	140 861	183 451
Households	448 519	916 831
Population size	1 313 103	2 463 651
Household size	[1 - 18] (3)	[1 - 16] (2)
Day cluster size	[1 - 25 339] (1)	[1 - 62 962] (2)
Home district size	[1565 – 2009] (2002)	[660 - 2 013] (2002)
Day district size	[1071 - 26 458] (2021)	[1370 - 62 962] (2002)

Algorithmic extension: sorting

The open source models [6, 10] process disease transmission by looping over all members of a cluster and if a member is infectious, to match them with all susceptibles. To reduce the total number of operations, we introduced a modified algorithm in which the members of a cluster are first sorted according health status before the infectious members are matched with the susceptible members. A newly infected member is moved ahead of the first susceptible. The member list obtains the following structure: First, recovered and infected (exposed and infectious) members and second, susceptible members. The following pseudo-code shows the sort algorithm for FRED and SID (the algorithm for FLUTE is structured analogously).

Transmission algorithm with sorting in FRED and SID:
1. **Loop** over all members $\{x\}$
2. **If** x is not susceptible
3. **If** index of $x >$ number infected + recovered
4. **Swap** x with first susceptible in the list
5. **Loop** over all non-susceptible members $\{x\}$
6. **If** x is infectious
7. **Loop** over all susceptible members $\{y\}$
8. **If** random number $< P_{tr}(age_x, age_y)$
9. **Start** disease in y

Parallelization: scheduling

The OpenMP API is often used for shared memory parallel programming in C/C++ [25]. In this programming model, subsets of a process are managed independently (=threads) and share a global address space of a single or multiple processors which they read and write asynchronously. For each cluster type (household, day district, ...) in an area, a person is a member of only one cluster. Therefore, clusters are stored per type so that these containers can be processed in parallel without synchronization. Parallel processing within one cluster would lead to synchronization overhead. The workload distribution over the threads can be static or dynamic [25]. With static scheduling, a fixed number of tasks are assigned to each thread. In dynamic scheduling, the workload is distributed over the idle threads until all tasks are done. We have used workloads in chunks of one and ten clusters.

Inputs and work environment

We used a 2×2 transmission matrix and assumed that the transmission probability (P_{tr}) is doubled for contacts between children (<18y) [20–22]. Similar to the literature [6, 10], we estimated the relationship between P_{tr} and the basic reproduction number R_0 by counting the number of secondary cases of one infected in a

complete susceptible population with seven P_{tr} values. Based on 4000 realizations with seven P_{tr}, we approximated R_0 by *exp(5507*P_{tr} -0.1911)*. The total run time depends on the clinical attack rate (AR, total fraction of the population initially at risk that got infected) and for this reason, we performed we performed benchmarks for a range of R_0 values (1.1, 1.25, 1.4, 1.8 and 3). Each simulation was performed for 100 days. To start the epidemic, we infected a random fraction of the population. After testing seeding rates of $1e^{-2}$, $1e^{-3}$, $1e^{-4}$ and $1e^{-5}$, we observed limited impact on the number of cases for these ranges and selected $1e^{-4}$ as baseline setting.

We included the pseudo-random number generator (PRNG) from an open source software package called TRNG [19, 26], a portable and highly optimized library of parallelizable generators. To prevent synchronization and latency, independent streams of random numbers are required for each thread. We used the robust and versatile "leapfrog" method where the PRNG sequence is distributed over *p* processes by calculating for draw *i* the *i*(p-1)*th number in the sequence. There are no recommendations to select PRNG seeds to obtain different stochastic results, except that those seeds have to be different. Therefore, the run index has been used to seed the PRNG.

An extended class diagram and the free open source code can be found in Additional file 1 and Additional file 2 respectively. Additional file 3 contains a user manual to make use of the project software. During development, we used the Google C++ Testing Framework [27] to perform detailed tests. These tests were applied in automated fashion with every change in the code base via a continuous integration server [28]. The Templatized C++ Command Line Parser library [29] was used to transfer configurations to the executable. The project-code is standard C++11 throughout, independent of external libraries and portable over all platforms that have a GNU compiler (version 4.8 or later) available.

Timings presented in this paper were obtained from benchmarks on a cluster with Intel® Xeon® E5-2680 v2 2.80 GHz CPU's (release Q3'13) from the HPC core facility CalcUA at the University of Antwerp. We confirmed our results with benchmarks on quad-core Intel® Xeon® W5580 3.2 GHz (release Q1'09) CPU's and AMD Opteron® 6274 CPU's. The GNU compiler (4.8) was used in release mode with compiler optimization "-O3". Additional file 4 contains more info on the hardware and extra results. The open source tool PerfExpert [30] was used for profiling, as installed on the CalcUA cluster.

We performed additional benchmarks to explore the effect of cluster size, dynamic clusters and increased model complexity on model performance. Methods and results can be found in Additional file 5.

Results and discussion

The number of infected people is the dominant factor in determining the computational workload and the required simulation time. Therefore, we needed to incorporate distinct epidemic curves in our benchmarks by using different R_0 values. Small deviations in the AR were observed for each R_0 as a result of different stochastic paths with and without the sort algorithm and given the different processing in FLUTE. To prevent stochastic fade-out, which is not appropriate for benchmarks, we used relatively high epidemic seeding rates to introduce new infected people in the population [12]. The benchmarks all report elapsed wall clock times as is appropriate for parallel programs. All results in this paper are based on mean timings from 10 runs with a different random number generator seed. With intervention strategies, we expect more stochastic fade-out and would require more realizations. Benchmarks are performed on idle computing nodes and results on other hardware can be found in Additional file 4.

Simulations with the basic models without concerns of the population order clearly required the longest run times. Figure 3 illustrates the total run time for FRED simulations with the Nassau population. Similar results were obtained with the other models (Additional file 4). We observed a large decrease in run time when the population is structured according to day cluster and household. The workload for a cluster of size *N* with *I* infectious and *S* susceptible members can be approximated by *N* health checks to select the infectious members + *I*(N)* health

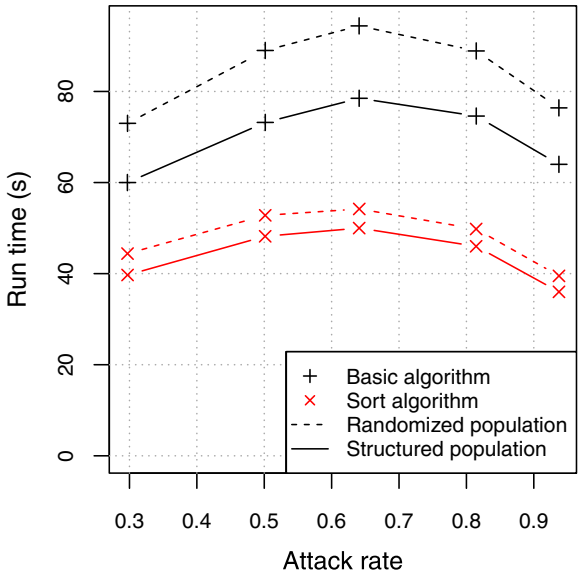

Fig. 3 Run time according to attack rate and population structure for Nassau simulations using FRED. Structured population: sorted according to day cluster (first) and household (second)

checks to select susceptible potential contacts + *I*S* random draws to match the infectious member with the susceptible members. The number of susceptibles decreases with each new case, which explains the decreasing curve in Fig. 3 for epidemics with high AR. Next, sorting cluster members on health status before processing disease transmission, had a large impact on the performance. The run times for Nassau were reduced with 26 % to 79 % compared to the basic models, depending on AR and population ordering. The algorithm with sorting has overhead because of swapping infected and healthy members but overcomes *I*(N)* health checks on susceptibility, which explains the reduced run times.

Given the impact of the AR on the simulation time, we needed to monitor the benchmarks closely. The stochastic transmission process is altered by the sorting algorithm, which has limited impact on the AR. Also, the population ordering determines the initial sequence of the cluster members and thus the random path of the simulator. Figure 4 presents the AR from 10 Nassau simulations using different models and population structures. The AR distributions were overlapping, which suggested similar transmission dynamics and approved run time comparisons. To validate the transmission model presented here, we performed simulations with the open source FRED software from Greffenstette et al. [10] using population data distributed with the source code for Allegheny, Pennsylvania. We observed ARs of ±68 % if $R_0 = 1.4$ and ill people could not stay home, which was close to our results.

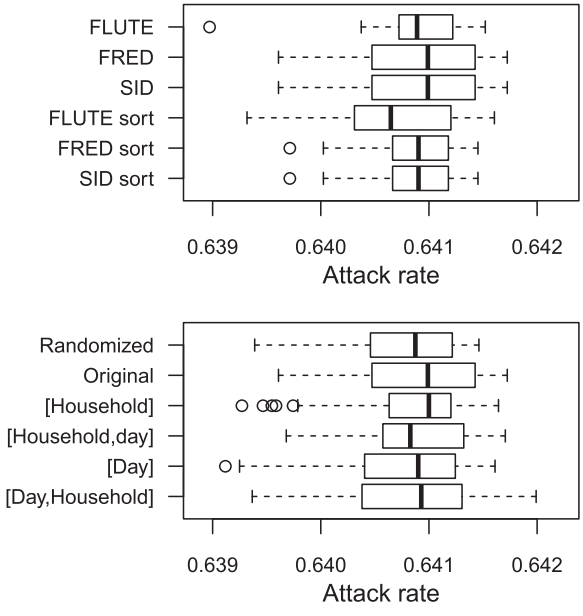

Fig. 4 Attack rates for Nassau simulation using $R_0 = 1.4$ and seeding rate = $1e^{-4}$ according to implementation (top) and population structure (bottom). Results from 10 simulations. The original population structure is used to compare implementations (top) and FRED to compare population structures (bottom). Box: upper and lower quartile, wisker: minimum and maximum excluding outliers, circle: outliers (>1.5x interquartile range), […]: structured population e.g., [Day, Household] represents populations sorted according to day cluster (first) and household (second)

The population ordering appeared to have a large impact on model performance. To examine the effect on an epidemic with $R_0 = 1.4$ (AR ±64 %), we used different versions of the population data with and without sorting according to household and/or day cluster. We repeated our benchmarks multiple times and did not observe large differences in ARs (Fig. 4). Table 2 presents the mean timings from multiple runs with each population ordering using the three basic models. The randomized populations gave the highest run times for all basic models. Using the original structure of the RTI population files slightly decreased the run time. Sorting the population on household ID improved the performance though most optimal was to sort the population on day cluster (first) and household (second). With this sorted population structure, we observed a reduction up to 59 % for FLUTE and FRED compared with the randomized population. The effect of the population structure was less for SID. The original open source FluTE model [6] uses a population sorted according to household. With our FLUTE implementation, we observed a decrease in run time of 20 % by using a population for Nassau sorted by day cluster and household. The population of the original FRED model [10] follows the structure of the RTI population files. A decrease of 6 % in total run time can be achieved with our FRED implementation by sorting the population file once. The impact of the population ordering was limited for the models with the sort algorithms.

The general trends from the Nassau simulations were also valid for Brooklyn. The improvement of the sorting algorithm ranged from 34 % to 63 %. For Brooklyn, we reduced the simulation time by sorting the population once with respectively 15 % and 19 % compared to

Table 2 Timings for Nassau simulations with different population structures

Population structure	FLUTE	FRED	SID
Randomized	108	91	95
Original RTI sequence	101	94	92
[Household]	94	86	87
[Household, Day]	93	80	87
[Day]	89	80	82
[Day, Household]	81	69	78

Results in seconds with $R_0 = 1.4$ and seeding rate = $1e^{-4}$ (AR = ±62 %). E.g., [Day, Household] represents populations sorted according to day cluster (first) and household (second)

the original FLUTE and FRED population. The highest improvement with the population structuring was 39 %. Table 3 presents the mean run times from 10 Brooklyn simulations with R_0 = 1.4. The ranking of the basic models based on total run time differed between Nassau and Brooklyn simulations due to the different population size and cluster size distribution. For the models with the sorting algorithm in the cluster class, the ranking was independent of the population structure. The extra effort to manage separated household and day clusters in FRED and SID improved the simulators' performance compared to the district-approach from FLUTE. The SID model with the sort algorithm performed best for all benchmarks, especially with the structured population.

On today's multi-core chips, memory access is a critical performance-limiting factor [31]. Therefore, we have analyzed software behavior and memory access patterns with a profiling tool for high-performance computing applications, PerfExpert [30]. We found that the function in *Cluster* to process disease transmission takes on average 98 % of the run time. Therefore, optimizations in this part of the code can have large impact. Since a member cannot be infectious and susceptible at the same time, it is not necessary to check whether a member tries to infect himself/herself. We observed that adding a simple comparison of two C++ pointers or two integer indices in FRED and SID respectively, increased the simulation time with 25 %. Table 3 presents a selection of the PerfExpert output. FLUTE had the highest penalty for branch instructions (if-then-else structures), which limits the CPU to pipeline instructions and to execute different stages (fetching, decoding, processing and store data) at the same time. A mispredicted branch instruction disturbs this optimization. FRED and SID required less cycles for branch instructions, especially with the sort algorithm. Sorting the cluster members before processing transmission also reduced the data access. Regarding the cache-coherency, we have observed that structuring the population according to the social contact clusters decreased the number of last level cache misses. The sorting algorithm disrupts the memory consistency by relocating references to cluster members. By comparing FRED and SID profiling results, we can confirm the targeted data management strategy from struct-of-array vs array-of-structs: the SID models have fewer last level cache misses.

Processing disease transmission requires many iterations over independent clusters and therefore seems suited for distributed programming. We observed that the effect of parallelization was dependent of the epidemic curve. Figure 5 presents differences in the speedup using FLUTE with 4 threads according to the AR and the epidemic seeding rate (= initial fraction of infected people). The different rates we used did not have impact on the total number of cases but only on the length of the initial phase with a small amount of infected clusters. Simulations with a high epidemic seeding rate and a large AR gave the best speedups using multiple threads. To illustrate the possibilities of parallelization, we compared simulation times using 1 to 20 threads for epidemics with R_0 = 1.4 and seeding rate = $1e^{-2}$ (AR ±64 %). Figure 6 presents the speedup for SID with basic and sort algorithm using a structured population according to day cluster and household. Similar results were obtained

Table 3 Profiling results for Brooklyn simulations

	FLUTE		FRED		SID	
	Basic	Sort	Basic	Sort	Basic	Sort
Randomized population						
- Branch instructions	0.29	0.14	0.23	0.07	0.18	0.07
- Data access	1.85	0.99	2.12	0.97	1.69	0.78
- LLC misses	0.14	0.14	0.27	0.48	0.13	0.29
- Run time (s)	229	114	237	103	222	102
[Day, Household] population						
- Branch instructions	0.26	0.14	0.22	0.07	0.17	0.07
- Data access	1.23	0.85	1.42	0.67	0.66	0.35
- LLC misses	0.05	0.07	0.12	0.25	0.04	0.12
- Run time (s)	168	103	188	94	147	88

Results with R_0 = 1.4 and seeding rate = $1e^{-4}$ (AR = ±62 %). All metrics, except run time, are given in LCPI: local cycles per instruction. LLC: last level cache. [Day, Household] populations are sorted according to day cluster (first) and household (second)

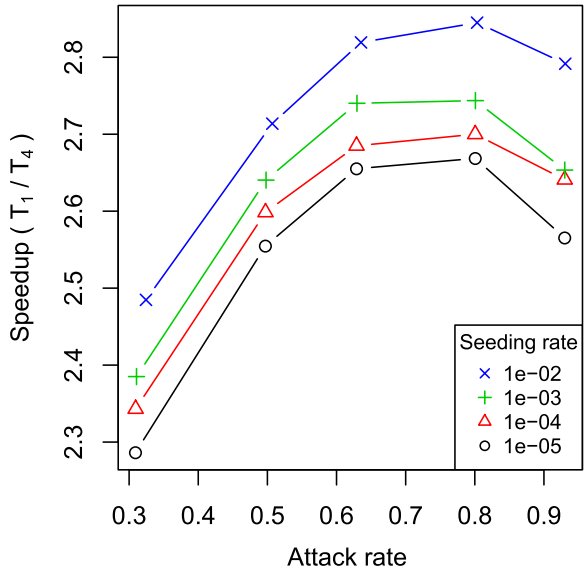

Fig. 5 Speedup according to attack rate and epidemic seeding rate using FLUTE (basic) with 4 threads. All simulations were performed with a structured Brooklyn population sorted according to day cluster and household using dynamic scheduling with workload chunks of 1 cluster

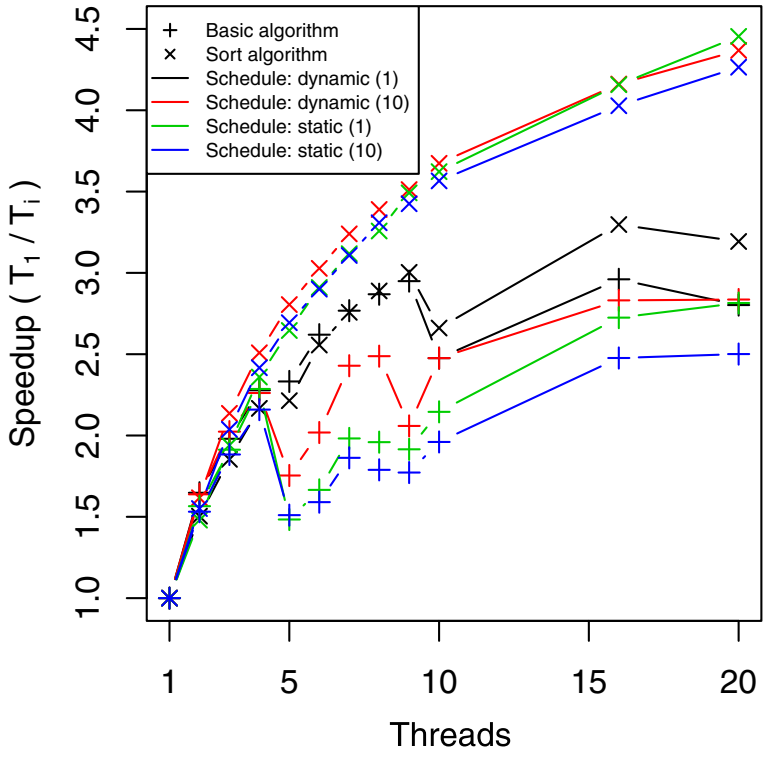

Fig. 6 Speedup according to thread number and scheduling for Brooklyn simulations using SID. Results are shown for the basic and sort algorithm with dynamic and static parallel scheduling using workload chunk size of 1 and 10 clusters. All simulations were performed with a structured population sorted according to day cluster and household with $R_0 = 1.4$ and seeding rate of $1e^{-2}$ (AR = ±64 %)

for the other implementations and using the randomized population, which can be found in Additional file 4. We observed good speedup for all models and scheduling options with 2 threads. With 3 or more threads, the added value of extra threads decreased due to memory bandwidth saturation. Making the clusters more self-contained by replacing the member references by actual person data would reduce this limitation although it requires much synchronization between the clusters and extra memory. All basic models had most benefit of dynamic scheduling with workload chunks of 1 cluster. With sorting, FRED and SID seemed to operate more optimally with static scheduling or dynamic scheduling with workload chunks larger than 1 cluster. For FLUTE, the dynamic scheduling with chunks of 1 cluster gave the best speedup. We tested the models on other hardware and observed similar results (Additional file 4).

By increasing model complexity, more different cluster types can be used and sorting the population might be less effective. If more person attributes are required for the disease transmission, co-locating these in a person object will be beneficial. On the other hand, the increased amount of person data will reduce the number of persons that fit in the high-speed cache, so more data needs to be fetched with higher latency. We explored these model aspects (Additional file 5) and observed that cluster size had a large impact on run time. Though, the differences regarding population sorting, model design (FLUTE, FRED and SID) and the sorting algorithm scaled with cluster size. To estimate the effect of dynamic clusters on model performance, we implemented a model with changing cluster membership over time. This way, the run time increased but the overall conclusions remained valid. Increasing model complexity by adding extra person attributes in FLUTE and FRED reduced the impact of the population sorting. The run times for SID remained constant if these attributes were not used, which confirmed the targeted data strategy of struct-of-array vs array-of-structs. The SID design became a disadvantage regarding model performance and workload for the programmer if these extra person attributes were involved in the transmission process.

Conclusion

ABMs offer a very powerful and flexible framework to analyze infectious disease transmission. Unfortunately they come with a large computational cost. Investing time in code optimization and adaptation to hardware innovations reduces time available for adding new features

although it can save much time during testing and in production.

We compared different ABM implementations for close-contact disease transmission models for two U.S. counties. Our ABM consisted of household, school, workplace and district clusters and people in a cluster can have social contacts and transmit an influenza-like disease. The transmission probability was assumed to be age dependent. We observed reductions up to 59 % by structuring the model population once according to the largest social contact cluster. Next, sorting the cluster members based on health status before processing disease transmission appeared also very beneficial for the model performance (reduction up to 79 % compared to the basic model).

Data movement and access require much more cycles than floating point operations and therefore data layout has impact on run time. We compared models that handle the population in large districts with models that also process the household and day clusters separately. The latter seemed beneficial for the performance especially in combination with the sorting on health status in the clusters. The storage of person data in separate containers instead of per person improved the data locality and cache-coherency and reduced modeling time. Models that sort cluster members on health-status before processing disease transmission are scalable with multiple threads if the epidemics have a limited initial phase. The parallel scheduling and workload chunk size had significant impact on the simulation time.

Increasing model complexity may reduce the impact of the population ordering. We describe the core of the simulator but more research is needed to assess the role of data layout and sorting algorithms together with mitigation strategies. Although improving data layout by using a separate container for each person feature might increase the model performance, it is counter intuitive for an ABM and requires extra effort from the modeler. The current software does not predict the workload before scheduling the chunks over multiple threads. We believe this scheduling would be a valuable extension to the parallel implementation because the cluster sizes and the amount of infected individuals per cluster can be very heterogeneous. In conclusion, large performance gains can be achieved with limited effort by structuring the population once, adding an algorithm that sorts by health status and selecting appropriate parallel settings.

Additional files

Additional file 1: Class diagram. Schematic overview of the project.

Additional file 2: Free open source code. Documented C++ code with Makefiles.

Additional file 3: User manual. User manual of the project software.

Additional file 4: Hardware specifications and extra results. Computer hardware details and more benchmarks for Nassau and Brooklyn simulations.

Additional file 5: Model exploration and validation. Benchmarks to assess the impact of model complexity on model performance.

Abbreviations

ABM: Agent-based model; AR: Attack rate; CPU: Central processing unit; PRNG: Pseudo-random number generator.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

LW, SS and JB conceived and designed the experiments. LW and SS performed the experiments. LW and ET analyzed the data. PB, NH and JB contributed materials/analysis tools. LW, SS, ET, PB, NH and JB wrote the paper. All authors read and approved the final manuscript.

Acknowledgements

We thank Geert Bostlap and Franky Backeljauw from the HPC core facility Calcua at the University of Antwerp for their inspiring lectures and support. LW is supported by an interdisciplinary PhD grant of the Special Research Fund (Bijzonder Onderzoeksfonds, BOF) of the University of Antwerp. SS is funded by the Agency for Innovation by Science and Technology in Flanders (IWT). NH acknowledges support from the University of Antwerp scientific chair in Evidence-Based Vaccinology, financed in 2009–2014 by a gift from Pfizer. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Author details

[1]Centre for Health Economics Research & Modeling of Infectious Diseases, Vaccine and Infectious Disease Institute, University of Antwerp, Antwerp, Belgium. [2]Modeling of Systems And Internet Communication, Department of Mathematics and Computer Science, University of Antwerp, Antwerp, Belgium. [3]Interuniversity Institute for Biostatistics and statistical Bioinformatics, Hasselt University, Hasselt, Belgium. [4]Department of Information Technology, Ghent University–iMinds, Ghent, Belgium. [5]HPC core facility CalcUA, Computational Mathematics, University of Antwerp, Antwerp, Belgium. [6]School of Public Health and Community Medicine, The University of New South Wales, Sydney, Australia.

Received: 21 November 2014 Accepted: 11 May 2015
Published online: 02 June 2015

References

1. Eubank S, Guclu H, Kumar VSA, Marathe MV, Srinivasan A, Toroczkai Z, et al. Modelling disease outbreaks in realistic urban social networks. Nature. 2004;429(6988):180–4.
2. Ferguson NM, Cummings DAT, Cauchemez S, Fraser C, Riley S, Meeyai A, et al. Strategies for containing an emerging influenza pandemic in Southeast Asia. Nature. 2005;437(7056):209–14.
3. Germann TC, Kadau K, Longini Jr IM, Macken CA. Mitigation strategies for pandemic influenza in the United States. Proc Natl Acad Sci. 2006;103(15):5935–40.
4. Degli Atti MLC, Merler S, Rizzo C, Ajelli M, Massari M, Manfredi P, et al. Mitigation measures for pandemic influenza in Italy: an individual based model considering different scenarios. PLoS ONE. 2008;3(3):1790.
5. Das TK, Savachkin AA, Zhu Y. A large-scale simulation model of pandemic influenza outbreaks for development of dynamic mitigation strategies. IIE Trans. 2008;40(9):893–905.
6. Chao DL, Halloran ME, Obenchain VJ, Longini Jr IM. FluTE, a publicly available stochastic influenza epidemic simulation model. PLoS Comp Biol. 2010;6(1):1000656.
7. Roche B, Drake JM, Rohani P. An agent-based model to study the epidemiological and evolutionary dynamics of influenza viruses. BMC Bioinformatics. 2011;12(1):87.
8. Laskowski M, Demianyk BC, Witt J, Mukhi SN, Friesen MR, McLeod RD. Agent-based modeling of the spread of influenza-like illness in an

emergency department: a simulation study. IEEE Trans Inf Technol Biomed. 2011;15(6):877–89.
9. Aleman DM, Wibisono TG, Schwartz B. A nonhomogeneous agent-based simulation approach to modeling the spread of disease in a pandemic outbreak. Interfaces. 2011;41(3):301–15.
10. Grefenstette JJ, Brown ST, Rosenfeld R, DePasse J, Stone NT, Cooley PC, et al. FRED (A Framework for Reconstructing Epidemic Dynamics): an open-source software system for modeling infectious diseases and control strategies using census-based populations. BMC Public Health. 2013;13(1):940.
11. Halloran ME, Ferguson NM, Eubank S, Longini IM, Cummings DA, Lewis B, et al. Modeling targeted layered containment of an influenza pandemic in the United States. Proc Natl Acad Sci. 2008;105(12):4639–644.
12. Willem L, Stijven S, Vladislavleva E, Broeckhove J, Beutels P, Hens N. Active learning to understand infectious disease models and improve policy making. PLoS Comput Biol. 2014;10(4):1003563.
13. Drepper U. What every programmer should know about memory. http://www.akkadia.org/drepper/cpumemory.pdf.
14. Giles M, Reguly I. Trends in high-performance computing for engineering calculations. Phil Trans R Soc A. 2014;372(2022):20130319.
15. Giles MB, Mudalige GR, Sharif Z, Markall G, Kelly PH. Performance analysis and optimization of the OP2 framework on many-core architectures. SIGMETRICS Perform. Eval. Rev. 2011;38:9–15.
16. Sutter H. The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobb s J. 2005;30(3):202–10.
17. Sutter H, Larus J. Software and the concurrency revolution. Queue. 2005;3(7):54–62.
18. Hayes B. Randomness as a resource. Am Sci. 2001;89(4):300–4.
19. Bauke H, Mertens S. Random numbers for large-scale distributed Monte Carlo simulations. Phys Rev E. 2007;75(6):066701.
20. Mossong J, Hens N, Jit M, Beutels P, Auranen K, Mikolajczyk R, et al. Social contacts and mixing patterns relevant to the spread of infectious diseases. PLoS Med. 2008;5(3):74.
21. Hens N, Ayele GM, Goeyvaerts N, Aerts M, Mossong J, Edmunds JW, et al. Estimating the impact of school closure on social mixing behaviour and the transmission of close contact infections in eight European countries. BMC Infect Dis. 2009;9(1):187.
22. Willem L, Van Kerckhove K, Chao DL, Hens N, Beutels P. A nice day for an infection? Weather conditions and social contact patterns relevant to influenza transmission. PloS ONE. 2012;7(11):48695.
23. RTI International: 2010 RTIU.S. synthetic population ver. 1.0. 2014 Downloaded from http://www.epimodels.org.
24. Wheaton W. 2010 U.S. synthetic population quick start guide. RTI international. 2014. Retrieved from http://www.epimodels.org.
25. Chapman B, Jost G, Van Der Pas R. Using OpenMP: Portable Shared Memory Parallel Programming vol. 10. Massachusetts, USA: MIT Press; 2008.
26. Bauke H, Brown WE, Fischler M, Kowalkowski J, Paterno M, Knuth DE, Press WH, Teukolsky SA, Vetterling WT, Flannery BP, et al. Tina's random number generator library. 2011. Retrieved from http://numbercrunch.de/trng/.
27. Google: C++ testing framework. 2014. Retrived from http://code.google.com/p/googletest/.
28. Smart J. Jenkins: the definitive guide. Sebastopol, California: O'Reilly Media; 2011.
29. Tclap. The templetized C++ command line parser library. 2014. Retrieved from http://www.tclap.sourceforge.net.
30. Fialho L, Browne J. Framework and modular infrastructure for automation of architectural adaptation and performance optimization for HPC systems. In: Supercomputing. Cham, Switzerland: Springer; 2014. p. 261–77.
31. Rane A, Browne J. Enhancing performance optimization of multicore/multichip nodes with data structure metrics. ACM Trans Par Comput. 2014;1(1):3.