Distributed Memory Parallel Implementation of Agent-based Economic Models

Maddegedara Lalith[1], Amit Gill[2], Sebastian Poledna[3], Muneo Hori[1], Inoue Hikaru[4], Noda Tomoyuki[4], Toda Koyo[4], and Tsuyoshi Ichimura[1]

[1] Earthquake Research Institute, The University of Tokyo, Bunkyo-ku, Tokyo, Japan
lalith, hori, ichimura@eri.u-tokyo.ac.jp
[2] Dept. of Civil Engineering, The University of Tokyo, Bunkyo-ku, Tokyo, Japan
gill@eri.u-tokyo.ac.jp
[3] International Institute for Applied Systems Analysis, Laxenburg, Austria
poledna@iiasa.ac.at
[4] Frontier Computing Center, Fujitsu Limited, Japan
inoue-hikaru, tomoyuki, toda.koyo@jp.fujitsu.com

Abstract. We present a *Distributed Memory Parallel* (DMP) implementation of agent-based economic models, which facilitates large-scale simulations with millions of agents. A major obstacle in scalable DMP implementation is to distribute a balanced workload among MPI processes, while making all the topological graphs, over which the agents interact, available at a minimum communication cost. We addressed this problem by partitioning a representative employer-employee interaction graph, and all the other interaction graphs are made available at negligible communication costs by mimicking the organizations of the real-world economic entities. Cache-friendly and low-memory intensive algorithms and data structures are proposed to improve runtime and scalability, and the effectiveness of each is demonstrated. The current implementation is capable of simulating 1:1 scale models of medium-sized countries.

Keywords: Agent-based economic models · large-scale simulations · MPI

1 Introduction

A recent trend in macroeconomics is to use Agent-based models (ABMs), that simulate the behavior of individual economic entities, often using simple decision rules. Macroeconomic ABMs relax two key assumptions at the core of the New Neoclassical Synthesis–the single, representative agent and the rational, or model-consistent, expectations hypothesis[1]. Since the financial crisis of 2007-2008, ABMs have been increasingly used[1]. In recent years several ABMs that depict entire national economies have been developed. The European Commission has in part supported this endeavor by funding large research projects like

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

CRISIS[5][2, 3], and EURACE[6] which is a large micro-founded macroeconomic model with regional heterogeneity[4–6]. Some of the recent ABMs aim to simulate entire national or regional economies, like the Eurozone, with comprehensive models that include each individual economic entity–each household, each firm, etc [4, 7]. Such 1:1 scale simulations with hundreds of millions of interacting agents are computationally demanding.

The lack of scalable High Performance Computing (HPC)–enhanced macroeconomic ABMs that are capable of simulating hundreds of millions of agents is a major barrier in the application of ABMs to simulate an entire national or a regional economy. A major difficulty in implementing scalable HPC extensions is the complicated interactions among agents which occur over several topological graphs. In shared-memory implementations, these interactions lead to race conditions, while in *Distributed Memory Parallel* (DMP) implementations with MPI (Message Passing Interface), each interaction generates one or more messages among MPI processes (e.g., CPU cores). A critical step in scalable DMP implementation is balanced distribution of the agents among MPI processes, while making all the interactions graphs available at least communication cost. Some parallel implementations circumvent this difficulty by making only one interaction graph available across MPI processes using non-scalable strategies[4]. Apart from such partial parallel implementations, we could not find any literature on complete parallel implementation of a macroeconomic ABM capable of simulating several million agents.

Inspired by the need of HPC-enhanced agent-based economic models, we developed a DMP code based on MPI. Poledna et al.'s[7] model is chosen as the base ABM since its features, like the credit-based market, expectations-based decisions by agents, production based on the goods purchased in previous period, etc., provide opportunities to attain higher parallel scalability. Further, their model is attractive from the application point of view: rich in level of details, parameters estimated from real data, realistic economic interactions, etc. To distribute a balanced workload, we partitioned the agents based on a representative employer-employee interaction graph, and all the other interactions were made available with a negligible amount of communications. Several methods of improving the serial performance of market interactions, which in turn reduce the load imbalances, are presented. The current implementation takes around 20 seconds to complete a single iteration of a 1:1 model of Austria ($\sim$10 million agents), making the simulation of a whole nation a reality. Though the current scalability is limited to several tens of MPI processes, it can be further improved by using accurate estimates of the amount of computations associated with each agent.

The rest of the paper is organized as follows. Section 2 discusses economic ABMs in a DMP perspective, and presents the challenges in scalable implemen-

[5] FP7-ICT grant 288501, `http://cordis.europa.eu/project/rcn/101350_en.html`

[6] FP6-STREP grant 035086, `http://cordis.europa.eu/project/rcn/79429_en.html`. See also: `http://www.wiwi.uni-bielefeld.de/lehrbereiche/vwl/etace/Eurace_Unibi/`

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

tations. Section 3 explains the details of the proposed DMP implementation scheme, providing details of domain decomposition, solutions to the challenges discussed in section 2, and improvements to some extensively used serial algorithms. Section 4 presents runtime statistics to demonstrate the effectiveness of solutions introduced in 3 and the parallel scalability.

For the sake of brevity, in the rest of the paper, the term *rank* or *MPI-rank* denotes an MPI process (e.g., CPU core), the term *parallel* implies distributed parallel implementation with MPI, and *message* or *communication* implies a point-to-point or collective MPI communication (e.g. `MPI_Send()` and `MPI_Recv()`, `MPI_Bcast()`, etc.).

2 Agent-based economic models: an HPC point of view

Though they vary in rules defining the agents' actions, most of the general economic ABMs have common agent types and interactions among them. Figure 1 shows a schematic diagram of a typical ABM. Depending on the simulated economic zone, there can be several tens to a few hundred industries, each consisting of a large number of firms. The regional economy is connected to the rest of the world through foreign buyers and sellers. The largest in number are household agents, consisting of workers, inactive households, and investors. The total number of firms is roughly 10% of the workers.

At a glance, simulating millions of agents by assigning equal-sized subsets of agents to each MPI process seems to be an ideal DMP application. However, scalable DMP implementation is a challenging task due to the complicated interactions among agents. Most of the interactions take place over either centralized graphs or dense graphs with random links. In DMP implementations, each link represents either one or two communications among ranks that possess the two agents defining the link. Unless the interactions are between random pairs of agents, all the messages between a pair of ranks can be combined and delivered in a single message. On the other hand, each interaction over random graphs requires an independent message per link, and an independent reply message if the interaction is bidirectional. The rest of this section gives the details of different interactions among the agents, and explains the difficulties in implementing a scalable HPC extension.

2.1 Interactions over centralized graphs

The interactions involving government and banks take place over centralized graphs (see Fig 2a). Of these, uni-directional interactions are easier to parallelize, while the bi-directional interactions introduce load imbalances due to the serialization of the related decision-making of the government or bank.

Interactions of households, firms, and banks with government All the workers, investors, firms, and banks pay various taxes to the government, while government pays various social benefits to the households. Tax paying is

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

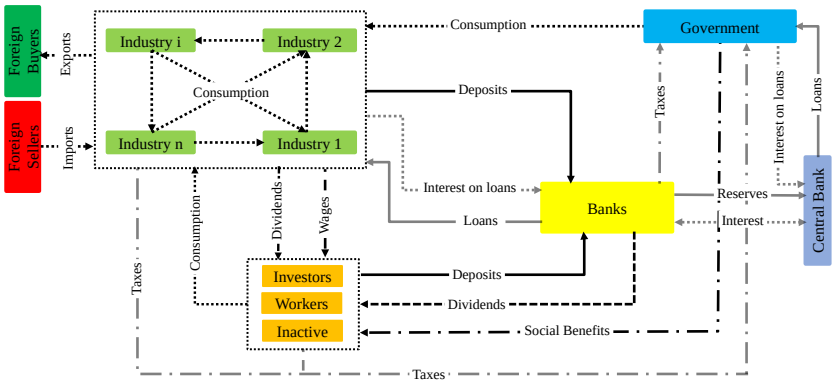
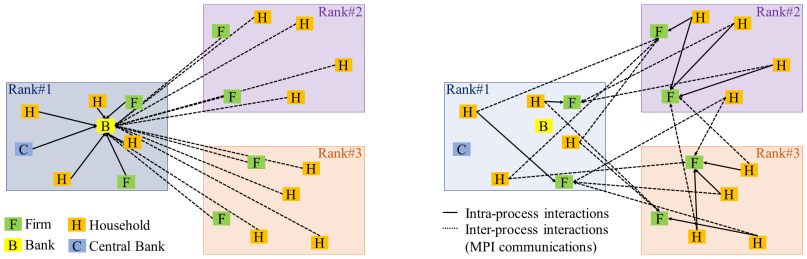


Fig. 1: Schematic diagram of a typical agent-based economic model. Each industry consists of a large number of firms.



(a) A centralized graph (b) A dense random graph

Fig. 2: Major sources of difficulties in shared- or distributed-memory parallel implementations. (a) hot spots like banks and government in centralized interaction graphs, (b) dense random interaction graphs like goods and job markets.

an unidirectional interaction, and can easily be parallelized by making each rank collect taxes and send them to the rank holding the government agent. On the other hand, paying social benefits is bi-directional, and involves three stages: first, the income states of households are gathered using `MPI_Gatherv()`; next, the social benefits to be paid to each household are *sequentially* calculated by the government agent; and finally, the social benefits are scattered to respective households using `MPI_Scatterv()`.

Interaction of households and firms with banks While depositing money is uni-directional, withdrawal and loan requests are bi-directional. The bi-directional interactions involve three stages, just like when the government is paying out social benefits: each MPI-rank sends households' or firms' requests to withdraw or borrow with `MPI_Gatherv()`; the bank sequentially processes each request; and responses are sent using `MPI_Scatterv()`.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

2.2 Interactions over dense random graphs

Unlike the interactions over centralized graphs, the interactions of goods and job markets take place over a set of dense random graphs: the connectivity of the graphs randomly changes in each time period to mimic the randomness in the goods and job market. These random graphs make it quite hard to implement scalable DMP extensions.

2.3 Interactions in the goods market

In each period, the consumers (i.e., firms, households, foreign buyers, and government sectors) visit random sets of sellers (i.e., firms and foreign sellers) from each industry to purchase the necessary consumption and capital goods. The number of sellers visited by a consumer agent is unpredictable, and depends on its current budget and the amounts available with each seller at the time it visits the seller. As a result, each buy-sell interaction involves independent messages among the ranks that possess the buyer and the seller. Being bi-directional, each visit to a seller involves three stages; a message to the rank possessing the corresponding seller, decision-making by the seller, and a reply message to the buyer. Developing a scalable shared- or distributed-memory parallel computing code to facilitate such random interactions among millions of agents is undoubtedly challenging.

A naïve solution is to introduce a queue at each seller, into which each consumer can submit a request with an MPI point-to-point communication, with the seller sending the reply message once the request is processed. Though logically correct, obviously the computational performance is much worse, even compared to the serial version.

2.4 Interactions in the labor market

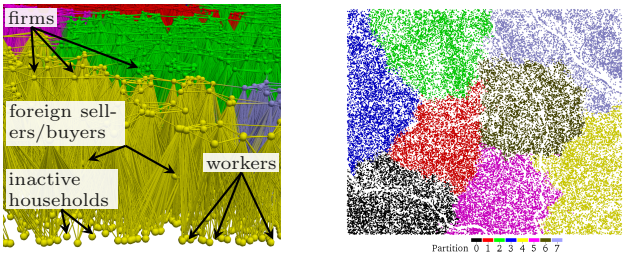
At the start of each period, firms hire a random set of unemployed workers, according to their labor requirements. When the available number of workers assigned to a rank is less than the total labor demand of the firms assigned to that rank, workers from the surrounding ranks have to be hired. Firms hire workers from a limited geographical extent to ensure real-life constraints, like travel time to work. If each firm independently seeks the required labor, it leads to a complicated situation similar to that of the goods market.

3 Distributed-memory parallel computing extension

This section presents the major steps involved in implementing an MPI-based DMP extension: partitioning the agents to assign balanced workloads to each rank; scalable solution for the difficulties discussed in the previous section; and some techniques for reducing the runtime of extensively called serial algorithms.

Though blocking MPI functions are used in the following text, corresponding non-blocking functions with user-defined MPI data types should be utilized to attain higher communication efficiency.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30



(a) Input graph to METIS (b) 8 partitions

Fig. 3: Input graph with 50,000 agents, and 8 partitions obtained with METIS. For clarity, different types of agents are drawn at three different elevations.

3.1 Domain decomposition

In order to distribute balanced computational workload among ranks, the agents are partitioned into mutually exclusive subsets. We use a graph representing the job market (i.e., employer-employee graph) in partitioning the agents, while taking different strategies to ensure the availability of rest of the topological graphs. The sections that follow explain how the remaining graphs, like buyer-seller, bank-customer, employer-employee, government-tax payers, foreign seller-local buyers, etc., are made available. What is desired are partitions with nearly equal sum of nodal weights (i.e., computational workload associated with agents) while minimizing the number of graph edges intersected by the partition boundaries, since each intersecting link gives rise to communications among ranks.

The graph to be partitioned is constructed by connecting each firm, f_i, with the closest n_i worker agents, where n_i is an approximate number of the labor requirement of f_i in the first period. To capture the reality, each agent is placed at the respective physical location, and the closest worker agents are identified using k-d tree. In order to ensure that the graph sufficiently reflects the employer-employee relations and has insignificant effects due to the presence of inactive households, foreign sellers, and foreign buyers, these three types of agents are connected to only one or two of the closest firms with low link weights. Further, each firm is connected to a few neighboring firms, with a low link weight, to reduce the possibility of producing a disconnected graph. The nodes of the graph (i.e., the agents) are assigned weights according to the amount of computations associated with each agent type. We partitioned this graph using METIS[8]. Figures 3a and 3b show a sample input graph and the 8 partitions generated.

Investors, banks, government, and central bank are not included in the above graph partitioning. Investors are assigned according to the number of firms in each partition. Government, bank, and central bank are assigned to the master rank (e.g., rank 0) to make them interact without any communications.

3.2 Scalable solutions for interactions over centralized graphs

By introducing representative agents of government, and banks in each rank, it is possible to almost eliminate the associated communications and equally

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

distribute the associated computational workloads, making the centralized interactions involving the government, and banks nearly perfectly scalable.

Interactions with government on centralized graphs The simple solution given in section 2.1 has quite a poor scalability since a single rank has to calculate the social benefits to be paid to millions of households, and involves messages of large volumes. A scalable solution is to introduce in each rank a *local government* agent which collects tax, calculates and pays the social benefits, and finally communicates the total tax collected and the social benefits paid to the master rank which holds the government agent. This simple solution is highly scalable since the computational workload is well distributed, all the bi-directional communications are eliminated, and the numbers of messages and volume of data communicated are drastically reduced.

Interactions with banks on centralized graphs A scalable solution for banks (see section 2.1) is to introduce into each rank *local bank branches,* that keep the accounts of all the customers in the corresponding ranks, and locally process all the deposit and withdrawal requests, thereby completely eliminating any messages and distributing the involved computations among the ranks.

When issuing loans to firms, a bank has to make sure that the total amount of the issued loans is less than a certain percentage of its total equity. That is, when processing m^{th} loan in period t, $L(t-1) + \sum_i^m \Delta L_i < \eta E(t)$, where $L(t-1)$ is the loans granted until the end of period $t-1$, ΔL_i is the i^{th} loan granted by the bank in period t, $E(t)$ is the equity of the bank, and η is the maximum allowable leverage for the bank. To strictly follow this condition at each local branch in a scalable manner, either a pre-estimated upper limit of the loans issuable at each local branch should be set or a relaxed condition like $L(t-1) < \eta E(t)$ should be used. We adopted the latter option since $\Delta L \ll L(t-1)$ is always valid. Both the options allow the local branches to issue loans without any need for contacting branches in other ranks, thereby eradicating any communications.

At the end of each period, local branches send a sum of their savings, loans, etc., to the main bank located in the master rank with a single `MPI_Gather()`. This drastic reduction in the number of messages and balanced distribution of computational workload make the introduction of local branches scalable.

3.3 Scalable solution for interactions over dense random graphs

Though, at a first glance, it seems impossible to implement a scalable parallel extension (see section 2.2), there are simple real-world solutions for goods and labor markets: sales outlets and recruitment agencies. We visit sales outlets, like supermarkets, instead of directly buying from producers. Scalable and least compromised solutions for the labor and goods market can be implemented by mimicking these real-world solutions.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

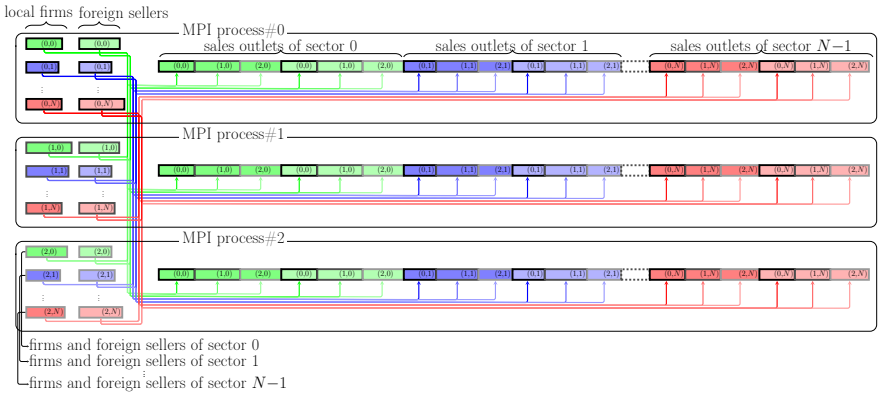


Fig. 4: Desired data layout of sales-outlets, and communications pattern.

Goods market We introduced *sales-outlets*, each of which sells products of a seller (i.e., a firm or a foreign seller). Each seller has one sales-outlet in each rank. The sellers communicate the total amount to be sold to all their sales-outlets using collective communications. The sales-outlet of the seller i in the rank r sets its amount to be sold based on the ratio $s_i^r(t-1)/\sum_{r=0}^{M} s_i^r(t-1)$, where $s_i^r(t-1)$ is the amount it sold in last period, and M is the total number of ranks. This requires an additional communication to gather $\sum_r s_i^r(t-1))$ to each sales-outlet. Once selling is complete, total sales and the demand of each seller are found by summing the corresponding data of its sales-outlets to master rank with `MPI_Reduce()`. Finally, the total sales and demands are scattered from the master rank to the corresponding parent sellers. This simple solution drastically reduces the number of MPI communications and completely eliminates any random communications, without compromising the buyer-seller interactions (i.e., buyers can use any random process within their respective rank).

While the introduction of sales-outlets solves the complicated communication problem, further planning is necessary to efficiently exchange the data among sellers and their outlets located in each rank. Figure 4 shows the desired layouts of the data of sellers and sales-outlets. The leftmost pairs of boxes indicate the sellers in each rank. The right-hand-side arrays of boxes indicate the corresponding sales-outlets. Using common MPI collective functions is not the most efficient since it involves at least $2I$ messages, where I is the number of industrial sectors; I calls to `MPI_Allgatherv()` to distribute products, and `MPI_Scatterv()` to collect sales information (e.g., sales and demand). This is a significant overhead[9] in simulating a country like Japan, which has 108 industrial sectors.

Some of communication overheads associated with I independent collective messages can be eliminated by combining [9] all the I using `MPI_Alltoallw()`. This more general collective function generalizes `MPI_Allgatherv()` by allowing to separately specify counts, displacements and MPI data types. Appropriately defining the MPI data types and displacements at the send and receive ends, `MPI_Alltoallw()` can be used to mimic `Allgatherw()` and `Scatterw()` operations and fetch and deliver the data in the desired format shown in Fig. 4.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

In the rest of the paper, this implementation of the goods market is referred to as `buy()`, and when referring to a specific consumer it is prefixed with the name of the consumer (e.g. households' `buy()`).

Labor market As mentioned in section 2.4, the labor market also poses a complicated communication problem similar to that of the goods-market. Though our partitioning scheme minimizes the number of labors to be imported/exported from/to neighboring ranks, the firms have to meet the required additional labor demands due to growth. We are exploring the advantage of mimicking recruitment agencies as a scalable solution. The recruitment agent in each rank keeps a record of which worker works where, and negotiates with the recruitment agents in neighboring ranks to allow excess workers of one rank to work in another.

3.4 Communication hiding

In order to attain higher parallel scalability, almost all the communications are overlapped with computations. Due to the space limitations, which communication is hidden behind which computations is not presented. Even though the blocking MPI functions are mentioned in the above explanations, we use corresponding non-blocking MPI functions (e.g., `MPI_Ibcast()`, `MPI_Ialltoallw()`, etc.). Whenever possible, we strive to post the non-blocking messages as soon as required data are available and finalize the receive right before the data are needed at the receiving ranks. Though not explicitly measured, this communication hiding must have made significant contributions to the scalability.

3.5 Serial performance enhancements

Several attempts were made to reduce the runtime of extensively called functions and thereby reduce the load imbalance among the ranks. As shown in the next section, the `buy()` function consumes about 95% of the total runtime. The runtime of `buy()` is significantly reduced by implementing a better performing function to draw random samples, and by using a data-oriented approach.

Draw from random distributions In most ABMs, consumer agents search for the best bargain. The consumers draw random samples from a distribution of sales-outlets of the industry from which they want to buy. The distributions are computed such that the outlets of the firms charging lower prices and/or producing larger quantities have a higher likelihood of being visited by customers.

To randomly select an outlet to visit, an agent generates a uniform random number r in the range $0 < r \leq \sum_1^n p_j$, and selects the corresponding outlet O_i from the distribution such that $\sum_1^{i-1} p_j < r \leq \sum_1^i p_j$, where n is the total number of active outlets and p_j is the likelihood of buying from active outlet O_j. Inclusion of the sold-out outlets in the distribution makes the consumer agents visit those pointlessly, thereby wasting a large number of CPU cycles.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

To eliminate this large wastage of CPU cycles, sales-outlets are removed from the distribution as soon as they are sold out. We refer this implementation of drawing from distribution as the *primitive* `draw_dist()` function.

The primitive `draw_dist()` is inefficient since deleting elements from the large arrays holding the distributions, is time-consuming. The efficiency of `draw_dist()` can be improved by disabling the sold-out outlets, instead of deleting. The sold-out O_k can be efficiently disabled by updating the distribution with $P_j = P_j - p_k$ for $k \leq j < N_s$, where N_s is the total number of sales-outlets in the industrial sector s, and $P_j = \sum_1^j p_l$. To skip this sold-out O_k, when drawing a random sales outlet from this new distribution, the smallest j which satisfies $r \leq P_j$ is chosen, for a given uniform random number r. This *improved* `draw_dist()` not only eliminates deletion from large arrays, but also eliminates a large number of conditional branching (i.e., `if` conditions). As shown in the next section, this improved `draw_dist()` significantly reduces the computation time.

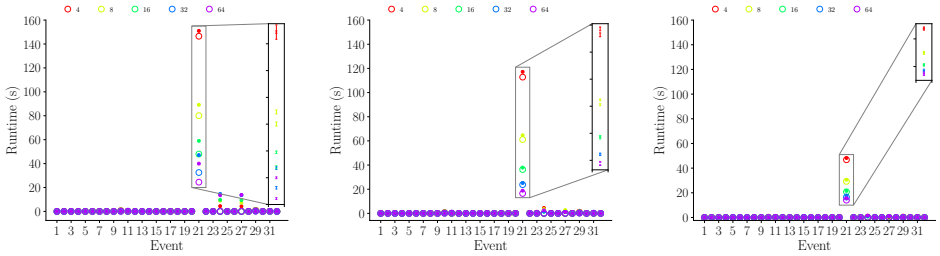
Data-oriented design In the `buy()`, millions of consumers visit hundreds of thousands of sellers from several tens of industries, one industry at a time. Obviously, this is highly memory-bound, and the size of the consumer objects and memory access patterns significantly influence the computational performance. In our `C++` implementation, a household agent consist of $12 + 2 \times I$ double-precision variables, where I is the number of industries. These bloated objects makes `buy()` of households an excessively memory-bound computation. Further, to provide fair buying opportunities, consumer agents are made to visit outlets in random sequences. Random access of a large array with bloated objects obviously has poor cache performance, leading to further performance degeneration.

In order to reduce the memory consumption and improve the cache performance of `buy()`, the amount of memory involved in `buy()` is drastically reduced by iterating though an array V, into which only 6 `buy()` related variables are copied from the household array. Further, the cache performance is significantly improved by randomizing V, instead of randomly accessing its components. Though lightweight, element-wise randomization of V is a memory-intensive task; hence we opt for block-wise shuffling. Since the size of V is large, block-wise shuffling is sufficient to produce fair opportunities in `buy()`. For efficient block-wise shuffling, a modified version of the Fisher-Yates algorithm[10] is used.

4 Computational performance

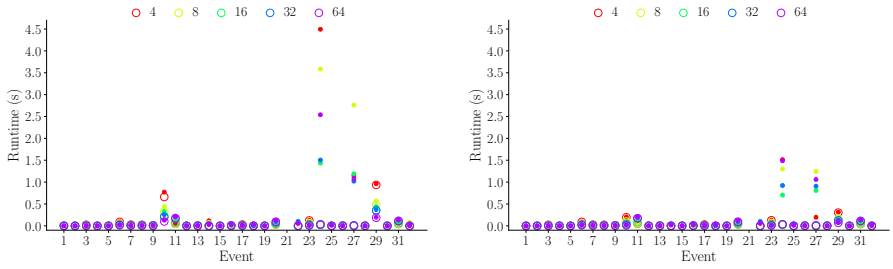
Based on the agent-based economic model by Sebastian et al.[7], we developed a distributed parallel code using `C++11` and MPI-3.2. The code is designed according to the domain decomposition and other techniques presented in the previous section. This section accesses the effectiveness of the improvements discussed in section 3.5, and the strong scalability.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30



(a) Primitive `dist_draw()` (b) Improved `dist_draw()` (c) Data-oriented `buy()`

Fig. 5: Mean runtimes (of 20 runs) of the 32 events. Zoomed views show the standard deviations.



(a) Improved `draw_from_dist()` (b) Data-oriented `buy()`

Fig. 6: Zoomed views of Figs. 5b and Fig. 5c.

4.1 Problem settings

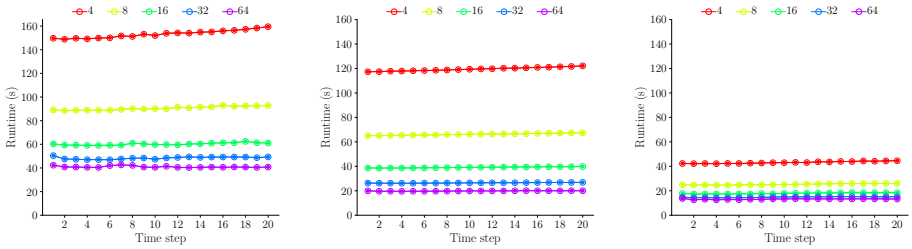
All the simulations were conducted with a 1:1 scale data set pertaining to the Austrian economy: altogether 10 million agents with 62 industries, 634,019 firms, 98,270 foreign sellers, 158,505 foreign buyers, 4,130,385 inactive households, 4,267,202 workers, 634,020 investors, one bank, central government, and central bank. The population is set to increase at the rate of 0.25% per time period.

Each simulation is conducted for 20 time periods with 4, 8, 16, 32, and 64 MPI processes in the ReedBush supercomputer of the Univ. of Tokyo. Each computing node consists of Intel Xeon E5-2695 v4 (2.1 GHz with 18 cores)×2 socket, and 256 GB memory with 153.6 GB/s bandwidth.

4.2 Significance of serial performance enhancements

Three sets of simulations were conducted to quantitatively estimate the contributions from the serial performance enhancements presented in section 3.5: primitive `draw_dist()`, improved `draw_dist()`, and data-oriented `buy()`. The main loop of the code comprises 32 events which include the basic events in the agent-based model, and the posting and finalizing of non-blocking messages.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30



(a) Primitive `dist_draw()` (b) Improved `dist_draw()` (c) Data-oriented `buy()`

Fig. 7: Runtime histories for the 20 simulated time steps.

4.3 Load balance and computational time

Figure 5 shows the time taken for each of 32 events of the main loop. The circles and dots indicate the mean values of the shortest and longest time taken by the respective ranks to complete each event. Differences between the corresponding circles and dots indicate the degree of load imbalance among ranks at each event.

The events 18, 19, 20, 24, 30, and 32 finalize non-blocking communications, while the events 2, 11, 12, 14, 15, 16, 22, 27, and 29 involve some computations and one or more non-blocking communications. Event 21, 24, and 27 are the `buy()`, finalization of the `MPI_Iallreduce()` to collect total sales to master rank, and finalization of the `IScatterw()` to inform the respective parents of the sales-outlets of those sales data (see section 3.3); events 22 and 25 post the corresponding messages.

Comparison of event 21 of Figs. 5a and 5b indicates that the primitive `dist_draw()` introduces a significant load imbalance, and its adverse effects are visible in events 24 and 27, which finalize the non-blocking messages related to `buy()`. Figure 5b shows that disabling of sold-out outlets, instead of deleting them from the distribution (see section 3.5), has not only eliminated a significant amount of load imbalances, but has also reduced the runtime of `buy()` by almost 25%. Further, it drastically reduces the imbalances in events 24 and 27.

Comparison of Figs. 5b and 5c shows that the lightweight data structure and cache-friendly data access introduced by the data-oriented improvements (see section 3.5) reduce the runtime by nearly 50%. The zoomed views of Fig. 6, shown in Figs. 5b and 5c, indicate that the data-oriented improvements of `buy()` have further reduced the imbalances of the events 10, 24, 27, and 29, which correspond to finalization of non-blocking messages. Event 29 updates the parent firms according to the sales information delivered by event 27; hence the reduction in event 29's runtime due to improvements in `buy()`. Since the main loop does not have any event that synchronizes the ranks, the load imbalances in `buy()` are transferred to the next time step, thereby introducing load imbalance to the events 10 and 11.

Event 16 and 19 correspond to the posting and finalizing `Iallgatherw()` with which sellers inform respective sales-outlets of the amount to be sold (see

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

Number of MPI ranks	Dist. updated at 1 + 0.25% growth rate		Dist. updated at 1 + 0.% growth rate		Dist. updated at 50 + 0.25% growth rate	
	Runtime (s)	Strong scalability	Runtime (s)	Strong scalability	Runtime (s)	Strong scalability
4	50.10		50.0		44.52	
8	31.97	78.4%	34.16	73.2%	25.97	85.7%
16	22.79	70.1%	22.87	74.7%	18.35	70.7%
32	19.12	59.6%	19.19	59.6%	15.23	60.2%
64	16.79	56.9%	16.74	57.3%	13.26	57.4%

Table 1: Average runtime and strong scalability with improved `dist_draw()` and data-oriented `buy()`. "Dist. updated at #" indicates that disabling is called when # outlets are sold-out. "#% growth rate" is population growth per period.

section 3.3). This is not only the most complex communication involved, but also the message carrying the largest data volume. Figure 5b and 5c clearly show that this, the most complex message, takes significantly less time and introduces no load imbalances, at least in relation to the imbalances introduced by `buy()`.

Figure 7 compares the runtime of the three cases considered above, for all the 20 time steps simulated. Figure 7a shows that not only is there a notable sudden variation and but also a notable increase in runtime with the time steps. Both these negative effects are induced by load imbalance in `buy()`. Though both of these effects diminish, the increase in runtime with the number of iterations is still present in Figs. 7b and 7c, due to the transfer of the load imbalance of `buy()` mentioned above.

4.4 Scalability

Table 1 shows the runtime and strong scalability[7] of the version with the improved `dist_draw()` and data-oriented `buy()` (Fig. 7c), under three different settings. In general, each setting produces a reasonable scalability up to 16 MPI ranks. As discussed, the main reason for lower scalability is the load imbalance in `buy()`. The three settings considered investigate other possible factors affecting runtime and scalability. A comparison of the 2^{nd} and 3^{rd} columns with the 4^{th} and 5^{th} shows that a gradual increase in the number of agents has a negligible impact on the runtime and scalability. On the other hand, a comparison of the 4^{th} and 5^{th} columns with the 6^{th} and 7^{th} shows that the time taken to disable the sold-out outlets by updating the cumulative probability distribution is considerable and heterogeneous among MPI ranks. Reducing the frequency of updating the probability distribution lowers total runtime by around 20%, although the scalability improvements are limited to a smaller number of ranks.

It is possible to further improve the scalability by eliminating the remaining load imbalances (see Fig. 6b) induced by `buy()`. When partitioning, we set the nodal weights approximately according to the number of floating point operations in the rules of each type of agent. This approximate estimation does not

[7] Strong scalability, a measure of how efficiently computational resources are utilized, is defined as $(T_m/T_n)/(n/m)$, where T_k is the runtime with k MPI ranks and $n \geq 2m$.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30

take details, like scale of production, number of workers, etc., into account. The load imbalance in `buy()` can be reduced by setting the nodal weights according to the measured runtime of each agent, thereby improving the parallel scalability.

5 Concluding remarks

A DMP implementation of an agent-based economic model capable of simulating medium-sized economies is presented. The major obstacle of distributing a balanced workload among MPI ranks, and facilitating all the agents' interactions at a minimum communication cost is addressed by partitioning a representative employer-employee interaction graph and mimicking real-life solutions. It is demonstrated that the runtime can be significantly reduced using a data-oriented design, and less memory-intensive, and cache-friendly algorithms. While the current implementation can simulate tens of millions of agents, larger simulations are possible through a better distribution of workload among MPI processes.

Acknowledgments This work was supported by JSPS kakenhi grant 18H01675. Parts of the results are obtained using K computer at the RIKEN Center for Computational Science, and the Reedbush supercomputer at the Univ. of Tokyo.

References

1. Haldane, Andrew G., and Arthur E. Turrell. An interdisciplinary model for macroeconomics." Oxford Review of Economic Policy, 34(1-2)(2018) 219-251.
2. Assenza, T., Delli Gatti, D., Grazzini, J.: Emergent dynamics of a macroeconomic agent-based model with capital and credit. J. of Economic Dynamics and Control 50(2015) 5-2
3. Klimek, Peter, Sebastian Poledna, J. Doyne Farmer, and Stefan Thurner. 2015. To bail-out or to bail-in? Answers from an agent-based model. J. of Economic Dynamics and Control, 50: 144-154.
4. Deissenberg, C., van der Hoog, S., Dawid, H.: EURACE: A massively parallel agent-based model of the European economy. Applied Mathematics and Computation 204(2008) 541-552
5. Fagiolo, Giorgio, and Andrea Roventini. Macroeconomic Policy in DSGE and Agent-Based Models Redux: New Developments and Challenges Ahead. J. of Artificial Societies and Social Simulation, 20(1)(2017).
6. Dawid, Herbert, and Delli Gatti, D.: Agent-Based Macroeconomics. Handbook of Computational Economics, ed. Cars Hommes and Blake LeBaron, 4(2018) 63-156.
7. Poledna, S., Hochrainer-Stigler, S., Miess, M.G., Klimek, P., Schmelzer, S., Sorger, J., Shchekinova, E., Rovenskaya, E., Linnerooth-Bayer, J., Dieckmann, Ulf, Thurner, S.: When does a disaster become a systematic event? Estimating indirect economic losses from natural disasters. arXiv:1801.09740
8. George Karypis and Vipin Kumar: A fast and highly quality multilevel scheme for partitioning irregular graphs. J. on Scientific Computing, 20(1), 359–392 (1999)
9. Balaji P., Chan A., Gropp W., Thakur R., Lusk E.: Non-data-communication overheads in MPI: Analysis on Blue Gene/P. LNCS, 5205, (2008)
10. Durstenfeld, R.. Algorithm 235: Random permutation. Communications of the ACM. 7(7): 420. (July 1964) doi:10.1145/364520.364540

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22741-8_30