Modeling 3D Voronoi diagrams using the visual parallel programming language VPPL

J. L. Quiroz-Fabián, G. Román-Alonso, M. A. Castro-García and M. Aguilar-Cornejo
Universidad Autónoma Metropolitana
México City, CDMX
Email: {jlqf,grac,mcas,mac}@xanum.uam.mx

***Abstract*—A Voronoi diagram is the partitioning of an area in regions. These regions are determined based on the proximity to a set of points in the area. Voronoi diagrams have many applications in different areas: Natural Sciences, Health, Engineering, Information Technology, among others. To build these diagrams, this work uses the VPPE parallel programming environment, and its VPPL visual language. With this environment, 3D Voronoi diagrams are build, using a drag-and-drop interface to select icons that allow the representation of a parallel program under the SPMD (Single Program Multiple Data) model. The results show the advantages of using the VPPL visual parallel programming against the usage of a textual parallel programming. Additionally, response times are shown for both, textual and graphic parallel programming, building different size Voronoi diagrams, using a cluster of computers.**

***Keywords*—Visual Languages, Graph Theory, Parallel Applications, Voronoi Diagrams**

I. Introduction

A Voronoi diagram is the partitioning of a area in regions. These regions are determined based on the proximity to a set of seed-points in the area. Each seed-point, X, generates a region whose points are closer to X than to any other seed-point. Voronoi diagrams have many applications in different fields: Natural Sciences, Health, Engineering, Information Technology, among others. Particularly, the modeling of 3D Voronoi diagrams with thousands of seed-points is very computing demanding, thus the development of efficient algorithms to satisfy this requirements is encouraged. In this work, a visual parallel program was proposed to build large 3D Voronoi diagrams, using VPPE (Visual Parallel Programming Environment) [1] [2]. VPPE is an IDE developed by our team; it mainly includes VPPL (a Visual Parallel Programming Language that consists of a set of icons to allow parallel programming through workflows design), a graphic editor, and an execution engine to use cluster resources. With VPPE, 3D Voronoi diagrams were obtained, through the workflow arrangement of several communication and SPMD (Single Program Multiple Data) processing icons of VPPL. The results show the advantages of programming with a visual parallel programming language, such as VPPL, compared to a textual parallel programming like MPI. Additionally, the performance evaluation is presented comparing the execution times obtained with the textual-sequential version and the proposed visual-parallel program, for different Voronoi diagrams sizes.

The structure of this document is as follows. Section II presents other works related to the generation of a Voronoi diagram by parallel computing. Section III presents the design and architecture of VPPE and VPPL. The VPPE workflow for generating a Voronoi diagram is presented in Section IV. A quantity comparison and the obtained performance by the VPPL workflow is shown in Section V. Finally, Section VI presents our conclusions and future work.

II. Related work

A Voronoi diagram is one of the most fundamental data structures in computational geometry and has been widely applied in variety of fields, either inside or outside computer science. Most effort has been directed towards developing efficient sequential or parallel algorithm by text based programming language. The first efforts are sequential or parallel algorithms that aim to decrease the computational complexity of n^2, $nlog^2n$, $nc^{\sqrt{logn}}$ to $nlogn$ using multiprocessors or distributed shared memory [3] [4] [5] [6]. With the emergence of GPU technology, new algorithms using tools like CUDA were proposed [7] [8] [9]. All these implementations require a good parallel programming knowledge for developers to generate a Voronoi diagram efficiently.

Although efficiency is very important (in Section V performance charts are presented), the aim of this work is to show the facility to generate Voronoi diagrams (or similar applications) by using a visual language. The representation of a visual parallel program has been done mainly using sequential diagrams, graphs, or workflows [10] [11] [12] [13] [14] [15] [16] [17][18]. However, most of the proposed visual languages or tools become complex when there are many edges among nodes, even more if different types of edges are used to specify communication and synchronization. Moreover, the representations of $time\ flow$ is poor or not considered. For instance, a graph does not specify what process starts execution or whether it gives a final result.

III. VPPE Architecture and language

The VPPE architecture is composed of four modules shown in Figure 1: A Developer Interface, a Translator, a Persistence Module, and the Execution Engine.

- The Developer Interface is composed of two elements: a Visual Language, VPPE Language (VPPL) and an Editor. VPPL provides developers with a set of icons

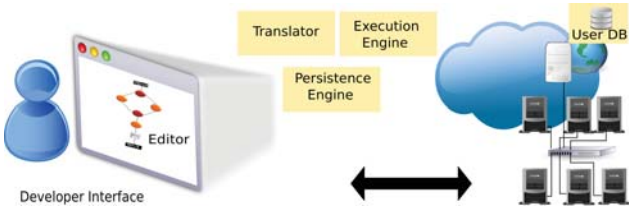

Fig. 1. VPPE development and execution environment.

and graphics structures to specify I/O communication or processing tasks within a workflow.

- Execution Engine: It is responsible for achieving MPI code generation (a program in MPI). This program is sent and executed on a previously defined cluster. During the execution, some workflow elements represent a group of processes. These processes make collective operations, such as Broadcast, Scatter and Gather.
- Translator: It translates a workflow into a text programming language. The current implementation generates source code in Java-MPI, being an appropriate tool for scalable systems.
- Persistence Engine: It includes a browser editor where the workflow is designed. This is saved on a disc as an XML file. The file can be loaded in the future, and the workflow can be modified/executed again.

A. VPPE Language: VPPL

The components of VPPL can be divided into four main groups: Workflow organization, Processing, Input/Output (I/O), and Communication icons (see Figure 2).

- Workflow structures (Figure 2, a): to represent the beginning and end of a workflow; to define all variables needed in a communication at any workflow point; to represent a lineal workflow structure; to represent a multiple branch parallel execution structure that specifies Multiple Program Multiple Data (MPMD) processing; a looping curved arrow structure to specify that a set of structures be executed repeatedly while some conditions holds.
- Processing icons (Figure 2, b): VPPL manages four general types of processing icons to represent: Sequential Processing, Single Program Multiple Data (SPMD) processing pattern, a pipeline pattern, and a Master-Slave pattern.
- I/O icons (Figure 2, c): Two icons represent the data input and output operations.
- Communication icons (Figure 2, d): VPPL supports three types of communication icons: unidirectional point to point (simple arrow), output, and input collective communication. The output collective communication structure can be of three types: broadcast (B-cast) allows a process to propagate the same data to a set of different processes; scatter (S-cast) allows a process to propagate different data to a set of different processes, and multiple-cast (BS-cast) allows a process to use both, a broadcast followed by a scatter operation. The input collective communication structure is used to specify that a sequential process gathers data (G-cast) from a set of other processes.

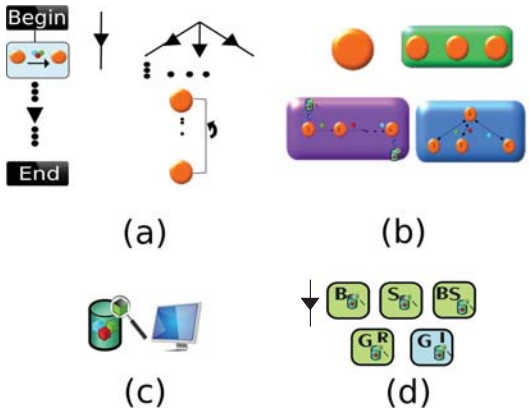

Fig. 2. VPPE structures and icons.

For more details about the VPPL, its icons and grammar see [1] and [2].

B. SPMD applications in VPPE

As described in section III-A, VPPE can manage various sequential processes and MPMD processes. As such, this rule is useful when a computation involves a relatively small number of processes, each running different code and processing different data. It could be used to specify an SPMD computation. However, the user would have to specify the same code for each instance of the same program. This is clearly unpractical if the application is to handle a high parallelism level which is common. For this reason we developed the SPMD pattern through a special icon. Figure 3 show the structure in order to manage this pattern.

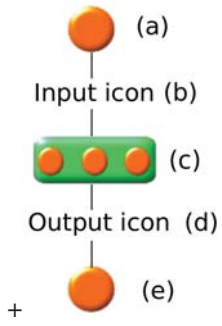

Fig. 3. SPMD in VPPE.

The icons (a) and (e) represent two processes, Begin and End. These processes are in charge of distributing and gathering data, respectively. The (b) icon represents (and is used to specify the type of) communication to distribute data using: scattering, broadcast and multiple-cast. The icon (c) is used to represent the SPMD processes — the user only needs to specify how many processes will be used and the code that all

them will execute. The icon (d) represents a gather operation by the End process (e) with the data sent by the SPMD processes (c).

IV. Generating Voronoi diagrams using VPPE

The VPPE aim is to make parallel programming easier, developers do not use textual operations for sending or receiving data. However, developers need knowledge about how partitioning the problem in order to get the expected outcomes. A parallel algorithm for generating 3D Voronoi diagrams consists of three main steps:

1) The sequential model.
2) Data partitioning.
3) Parallel process.
4) Data gathering.

A. The sequential model

For us a 3D Voronoi diagram is represented as a cube of NxNxN of integer coordinates. The points in a cube are classified in two groups: seed-points and the free-points. Each seed-point has associated a color and figure (box, circle, triangle and star). Free-points (little circles) look for the closest seed-point using then Euclidian distance, and take its color and figure. In Figure 4 a 3x3x3 diagram is showed.

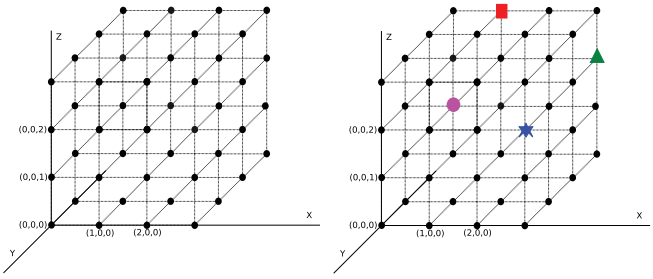

Fig. 4. Model for generating a Voronoi diagram.

A set of points are randomly selected and they are assigned like seed-point. They take the colors: red (box), blue (star), green (triangle), and fuchsia (big circle) . The others points are free-points (black points). All free-points will take a color and a Voronoi diagram will be generated (Figure 5). In this example, the generated 3D Voronoi diagram is not of 3x3x3 dimensions in order to generated a solid diagram. The dimension of this diagram is 400x400x400 (Figure 5, right).

B. Data Partitioning

The partitioning consists of dividing the x-axis, y-axis and z-axis in k parts. The number of divisions (parts) in each axis can be different. This allows to get a set of cubes (sub-cubes). Each sub-cube is represented by a tuple: (x_{nitial}, x_{end}, y_{nitial}, y_{end}, z_{nitial}, z_{end}), and a sub-cube can be independently processed by a process. Figure 6 shows an example where the partitioning of each axis is in two parts generating eight sub-cubes.

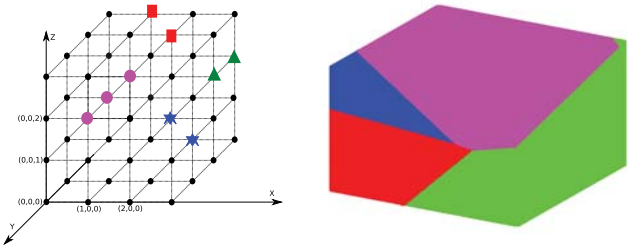

Fig. 5. Generating a Voronoi diagram.

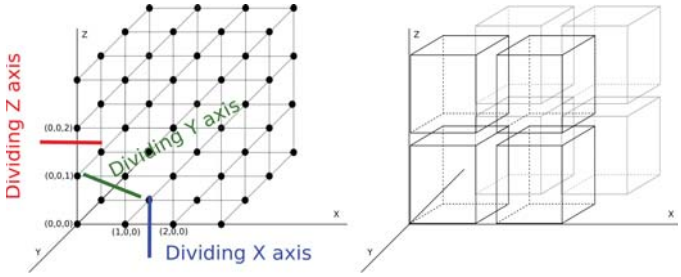

Fig. 6. Data partitioning in the Voronoi solution.

C. VPPE Voronoi Algorithm

Figure 7 shows the VPPE workflow for the 3D Voronoi diagram generation. Each blue circle allows to describe the flow and elements that integrate this workflow.

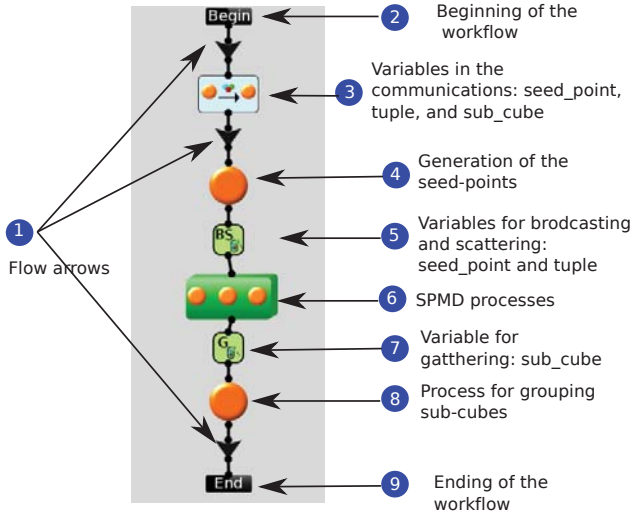

Fig. 7. Workflow for generating Voronoi diagram in parallel.

The circle 1 define the workflow's flow. The number 2 is the beginning of the workflow. The number 3 contains the variables used for communications. In our case we need three matrix:

1) *seed_point[N][3]*: a matrix for N seed-points randomly generated,
2) *tuple[P][6]*: a matrix with the values (x_{nitial}, x_{end}, y_{nitial}, y_{end}, z_{nitial}, z_{end}). P represents the number

of processes = number of sub-cubes,

3) *sub_cube[X_part×Y_part×Z_part]*: an sub-cube (represent as an simple array) that defines part of the resulting Voronoi diagram.

The number 4 defines a sequential process that generates the seed-points (initializes the *seed-point* matrix) and sets the limits of each *sub-cube* (initializes the *tuple[P][6]* matrix). The Matrix *tuple* will be scattered to the SPMD processes following the model presented in Figure 6. The number 5 contains the variables in the communications. In this icon is specified the matrix *seed_point* that is transferred by a broadcast operation and the matrix *tuple* using a scatter operation. The number 6 has the parallel processing by SPMD processes. Each SPMD process receives a copy of the *seed_point* matrix and the limits of its sub-cube. With this data, each SPMD process generates a Voronoi diagram in *sub_cube*, which is part of the final results. In this step, a SPMD process can practically use the sequential solution (code) of the problem. At this time, the generated *sub-cube* can not be transferred to another process for its gathering because VPPE only allow to send arrays in 1 or 2 dimensions, Developers have to translate the *sub_cube* in an array in order to be sent. The number 7 specifies what data will be sent to a sequential process for its group, in our case, part of the Voronoi diagram. Number 8 defines a sequential process grouping the sub-cubes as a integrated whole. Finally, the number 9 represent the end of the workflow.

V. Results and evaluation

Visual programming using VPPE (and its language VPPL) is simpler that the Text-based programming (TBP) for many applications. This is due to its icons for representing communication and synchronization operations. For generating Voronoi diagrams with large number of points using VPPE, Developers have to use these icons and connected them. By contrast, using paralelims through TBP they have to use a shared or distributed memory with its respective operations, for example, in the case of distributed memory they explicitly have to use send, receive, broadcast, scatter, among others.

In this Section a qualitative and performance comparison for generating Voronoi diagrams using VPPE and TBP is presented. The aim of the first is to demonstrate the benefits (facility) of using VPPE for generate a parallel application, and the aim of the second is to show the reduction of response time respect to the sequential version in TBP, where Developers do not use new instructions or operations as it is done in parallel programming.

A. Qualitative comparison: VPPL vs TBP

Developing a parallel application in distributed memory for generating Voronoi diagrams requires employs a set of operation for partitioning the area and to obtain the final (global) result. A set of these main operations are: processes creation, processes identification, processes group management, processes interaction (communication and synchronization). In the Table I a qualitative comparison assuming a MPI implementation in TBP and another in VPPE is presented in order to show the benefits of using the latter.

Characteristic	**TBP**	**VPPL**
Processes creation	In TBP is necessary instructions for processes creation and destruction, for example, *mpirun*, *MPI_Init* and *MPI_Finalize*.	Using VPPE/VPPL only is necessary to drag the SPMD structure and set the number of processes. The beginning and end icons are for default.
Processes identification	Its is necessary to know who is the process that partitions the points and who gathers the partial results. Developers have to use MPI functions such as *MPI_getRank* and *MPI_getSize*:	It is not necessary, each roles is setted for the SPMD structure.
Processes group management.	Not all processes have the same function, Developers have to use operation such as: *MPI_Incl*, *MPI_create* and *MPI_getGroup* (using an array with the process ids that belong to the group) if group creation is used.	The structure SPMD use implicitly two process groups: the first sequential process, the SPMD processes, and the last sequential processes and the SPMD processes.
Communication	For the partitioning is necessary to use operation point to point, send and receive, or collective operation such as broadcast, for sending seed-points, scatter, for partitioning the area, and gather, for obtaining the global result (diagram). In MPI these operation are *MPI_send*, *MPI_receive*, *MPI_bcast*, *MPI_scatter*, and *MPI_gather*.	Only it is necessary icons for collective communication. An BS and G icon. In the BS icon the variable name with the seed-points is necessary for broadcasting and the name of the arrays with the ranges of each process for scattering. In the G icon, only the name of the variable for gathering the partial results.

TABLE I. Qualitative comparison between TBP and VPPL.

The parallel solution with VPPE is simpler than TBP. Intuitively, this is because VPPE visual icons correspond to higher-level abstractions of processing tasks that hide many programming details in parallel computing.

B. Performance evaluation

A sequential application for generating a Voronoi diagram was developed. Using this application as base the VPPE workflow was generated, and in this section we show the performance of both versions to get a view of how good is the code generated by VPPE. We present first the experimental platform, and finally the results.

1) Experimental platform: The experimental platform is a cluster composed of 136 cores in 13 nodes; the hardware specifications and the operating system are shown in Table II. All nodes are connected through a Gigabit Ethernet switch.

Node	Processors/Cores	Frequency	Memory	Operating system
node 1-6	8	2.4GHz	8GB	Centos 7.2
node 7-11	8	2.3GHz	8GB	Centos 7.2
node 12	24	2.3GHz	8GB	Centos 7.2
node 13	24	2.3GHz	8GB	Centos 7.2

TABLE II. Experimental platform characteristics.

The software specifications are: gcc compiler version 4.8.5, OpenMPI version 1.10.2, and SDK java 1.8.0_25.

2) Results: A sequential application for generating Voronoi diagrams was programmed from scratch using Java. Two sizes of diagram were considered: 400^3 and 600^3. Both applications use 1000 seed-points. In the VPPE version the axis x and y

are divided in two parts and the division in z depends of the size diagram and if it is multiple of 4, this last for the number of partitions (four) obtained of the division in x and y (see Figure 6, right). The sub-cubes generated by the division in z are processed by SPMD processes.

The execution times obtained in the generation of a Voronoi diagram of 400x400x400 are shown in Figure 8. The x axis describes the number of SPMD processes: 4, 8, 16, 20 and 40, and the y axis the execution time (in milliseconds). The first point in x (x=1) represents the sequential program. We observe that the performance of VPPE is better than the sequential version and its time decreases when the number of SPMD processes increases. However, generating these diagrams using VPPE is very simple respect to parallel text based programming.

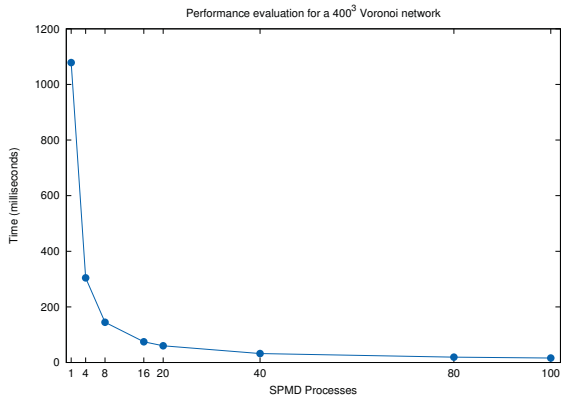

Fig. 8. Execution time for generating a Voronoi diagram of 400^3.

The execution times obtained in the generation of a Voronoi diagram of 600x600x600 are shown in Figure 9. As in the previous case, the first point in x represents the sequential program and the others the numbers of SPMD processes: 4, 8, 12, 20, 24 and 40. Similar as in the last graph, the performance of VPPE is better when more SPMD processes are used. For this size, a 3D Voronoi diagram is shown in Figure 10.

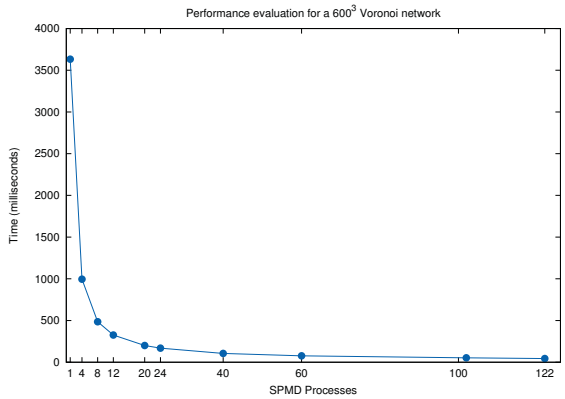

Fig. 9. Execution time for generating a Voronoi diagram of 600^3.

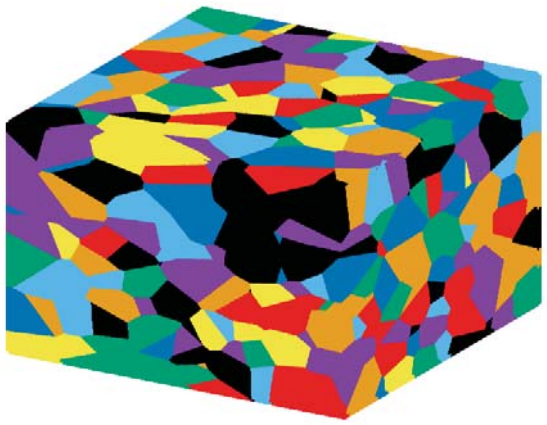

Fig. 10. Voronoi diagram of 600x600x600 and 1000 seed-points.

VI. Conclusion and Future work

We have presented a proposal to develop Voronoi diagrams using VPPE, a visual parallel programming environment, and its visual language VPPL. The programming of this applications was quasi sequential. The objective of the VPPE is to reduce the complexity of explicitly controlling the communication and synchronization of parallel applications. VPPL language simplifies the development of parallel applications, starting from a sequential version.

Future work includes the extension of VPPE to be improved in several ways.

- Integration of new data partitioning. This could support applications such as the presented in this paper for generating Voronoi diagrams, where data can be distributed automatically, in a transparent way.
- Transfer of new data types: For example, an array in 3D and avoid data conversion by the Developer.
- The execution in different architectures (multi-core or GPU).
- Performance comparison between the generated code by VPPE and a program written by an expert in parallel computing.
- Inclusion of load distribution algorithms in the architecture.

References

[1] J. L. Quiroz-Fabián, G. Román-Alonso, M. A. Castro-García, M. Aguilar-Cornejo, and J. Buenabad-Chávez, "A graphical language for development of parallel applications," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'13*, ser. PDPTA'13. Las Vegas, Nevada, USA: WorldComp 2013 Proceedings, 2013, pp. 672–678.

[2] J. L. Quiroz-Fabián, G. Román-Alonso, M. A. Castro-García, J. Buenabad-Chávez, and M. Aguilar-Cornejo, "A graphical environment for development of mpi applications," in *Proceedings of the 21st European MPI Users' Group Meeting*, ser. EuroMPI/ASIA '14. New York, NY, USA: ACM, 2014, pp. 125:125–125:126. [Online]. Available: http://doi.acm.org/10.1145/2642769.2642793

[3] S. Fortune, "A sweepline algorithm for voronoi diagrams," in *Proceedings of the Second Annual Symposium on Computational Geometry*, ser. SCG '86. New York, NY, USA: ACM, 1986, pp. 313–322. [Online]. Available: http://doi.acm.org/10.1145/10515.10549

[4] C. Levcopoulos, J. Katajainen, and A. Lingas, "An optimal expected-time parallel algorithm for voronoi diagrams," in *SWAT 88*, R. Karlsson and A. Lingas, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 190–198.

[5] R. Cole, M. T. Goodrich, and C. Ó. Dúnlaing, "Merging free trees in parallel for efficient voronoi diagram construction," in *Automata, Languages and Programming*, M. S. Paterson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 432–445.

[6] S.-Q. Xin, X. Wang, J. Xia, W. Mueller-Wittig, G.-J. Wang, and Y. He, "Parallel computing 2d voronoi diagrams using untransformed sweepcircles," *Computer-Aided Design*, vol. 45, no. 2, pp. 483 – 493, 2013, solid and Physical Modeling 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010448512002369

[7] T. Honda, S. Yamamoto, H. Honda, K. Nakano, and Y. Ito, "Simple and fast parallel algorithms for the voronoi map and the euclidean distance map, with gpu implementations," in *2017 46th International Conference on Parallel Processing (ICPP)*, Aug 2017, pp. 362–371.

[8] J. Toss, J. Comba, and B. Raffin, "Parallel voronoi computation for physics-based simulations," *Computing in Science Engineering*, vol. 18, no. 3, pp. 88–94, May 2016.

[9] J. Toss, J. L. D. Comba, and B. Raffin, "Parallel shortest path algorithm for voronoi diagrams with generalized distance functions," in *2014 27th SIBGRAPI Conference on Graphics, Patterns and Images*, Aug 2014, pp. 212–219.

[10] M. Al-Mulhem and S. Ali, "Visual occam: Syntax and semantics," *Computer Languages*, vol. 23, no. 1, pp. 1 – 24, Apr. 1997.

[11] A. Beguelin and J. J. Dongarra, "Graphical development tools for network-based concurrent supercomputing," in *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, ser. Supercomputing '91. New York, NY, USA: ACM, Nov. 1991, pp. 435–444.

[12] S. Böhm, M. Běhálek, and O. Garncarz, "Developing parallel applications using kaira," in *Digital Information Processing and Communications: International Conference 2011*, ser. ICDIPC 2011. Berlin, Heidelberg: Springer Berlin Heidelberg, July 2011, pp. 237–251.

[13] Y. Ce, X. Zhen, S. Ji-zhou, M. Xiao-jing, H. Yan-yan, and W. Hua-bei, "Paramodel: A visual modeling and code skeleton generation system for programming parallel applications," *SIGPLAN Not.*, vol. 43, no. 4, pp. 4–10, Apr. 2008.

[14] Chan, J. N. Cao, A. T. S. Chan, and M. Y. Guo, "Programming support for MPMD parallel computing in ClusterGOP," *IEICE Transactions on Information and Systems*, vol. E87D, no. 7, pp. 1693–1702, Jul. 2004.

[15] D. Ferenc, J. Nabrzyski, M. Stroinski, and P. Wierzejewski, "Visual mpi - a knowledge-based system for writing efficient mpi applications," in *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. European PVM/MPI 1999. London, UK: Springer-Verlag, Sep. 1999, pp. 257–264.

[16] S. Nenad and Z. Kang, "Visual programming for message-passing systems," *International Journal of Software Engineering and Knowledge Engineering*, vol. 9, no. 3, pp. 397–423, Aug. 1999.

[17] Z. Farkas and P. Kacsuk, "P-grade portal: A generic workflow system to support user communities," *Future Generation Comp. Syst.*, vol. 27, no. 5, pp. 454–465, May 2011.

[18] P. Newton and J. C. Browne, "The code 2.0 graphical parallel programming language," in *Proceedings of the 6th International Conference on Supercomputing*, ser. ICS '92. New York, NY, USA: ACM, Jul. 1992, pp. 167–177.