Accelerating a Virtual Ecology Model with FPGAs

Julien Lamoureux, Tony Field, and Wayne Luk
Deptartment of Computing
Imperial College, UK
{jlamoure, ajf, wl}@doc.ic.ac.uk

***Abstract*—This paper describes the acceleration of virtual ecology models using field-programmable gate arrays (FPGAs). Our approach targets models generated by the Virtual Ecology Workbench (VEW); an existing tool used by biological oceanographers to build and analyze models of the plankton ecosystem in the upper ocean. Depending on the plankton study and required level of detail, the logic, memory, and data transfer requirements of the generated models can vary significantly. Using FPGAs, hardware implementations can be customized to the specific requirements of the ecological system under study and provide significant speed-ups compared to software implementations. This paper describes a framework for maximizing the speedup of VEW generated models implemented on FPGA-based acceleration platforms and then describes the implementation of a typical VEW generated model to validate the framework and demonstrate that significant speedups are possible. Based on timing and area estimates from a commercial synthesis tool, the example model implemented on a Celoxica RCHTX acceleration board featuring a Xilinx Virtex-4 FPGA performs 39 times faster at 150 MHz than the software implementation on an AMD Opteron 2200 series CPU at 1.0 GHz.**

***Index Terms*—ecology modeling, hardware acceleration, FPGA**

I. Introduction

Significant improvements in FPGA size, speed, and storage capacity have made them suitable for accelerating a wide range of computationally intensive applications. Significant speedups achieved using FPGAs to accelerate cryptography, sparse matrix-vector multiplication, Viterbi decoding, and financial computing systems have been reported in recent literature [1], [2], [3], [4]. As an example, the study described in [2] demonstrated a 2 times speedup for floating-point sparse matrix-vector multiplication (a computational kernel at the heart of many scientific computing applications) implemented on a Virtex II FPGA compared to the fastest single processor system at the time and even greater speedups for multi-FPGA systems (compared to multi-processor systems). These speedups can be attributed to the high on-chip and chip-to-chip bandwidth provided by FPGAs.

This paper proposes the use of FPGAs to accelerate ecology models. Specifically, it focuses on models generated using the Virtual Ecology Workbench (VEW), which is used by biological oceanographers to build and analyze models of the plankton ecosystem in the upper ocean [5]. VEW models have been used to study competition in the plankton ecosystem [6], bio-optical feedback in ocean color [7], and the effect of weather on juvenile recruitment in fisheries [8]. They also have potentially important applications in understanding the role of marine plankton in the regulation of atmospheric CO_2 and, subsequently, climate.

The logic, memory, and communication requirements of the ecological models generated using VEW can vary significantly depending on the nature of the study. Using FPGAs, hardware implementations can be customized to the specific requirements of the ecological system under study and provide significant speed-ups compared to software implementations. The acceleration of these types of models is important because it allows the oceanographers to build more sophisticated models, run longer simulations, and/or perform more experiments which leads to a better understanding of ecological systems. The main challenge with using FPGAs to accelerate VEW models is mapping the model to the underlying acceleration platform. A tool that performs this mapping automatically is needed to make this type of acceleration feasible for oceanographers. This paper takes the first steps in this direction. Specifically, the following contributions are made:

1) A set of *ecology parameters* and *platform parameters* that determine the performance of an implementation are identified in Section III. The ecology parameters describe attributes specific to the ecosystem model. For example, the number of bytes needed to describe plankton agents affects the time it takes to transfer data between the host and the FPGA. Similarly, the platform parameters describe attributes specific to the acceleration platform. For example, the size of the FPGA determines the number of computational kernels that can be implemented in parallel. This directly affects the time required to process the entire plankton population.
2) A framework that estimates the performance of an implementation for given sets of model and platform parameters is described in Section IV. This framework is useful because it provides high-level system performance estimates, gives insight into possible performance bottlenecks, and helps in choosing the appropriate platform and implementation for a specific ecosystem model.
3) An example VEW generated model is implemented on an FPGA-based platform to show that custom FPGA implementations can produce significant speedups and to validate the performance estimation framework (Section V). The example implementation is 39 times faster (at 150 MHz) than the existing software implementation (at 1.0 GHz).

II. Background

The Virtual Ecology Workbench (VEW) [5] is used by biological oceanographers to build and analyze *individual-based* models of the plankton ecosystem in the upper ocean [9]. The individual-based approach is important, as it avoids the chaotic instabilities often observed in *population-based* approaches to ecosystem modeling [10]. Such models are used to explore the emergent properties of large numbers of individual microscopic organisms and the nature of the various feedback processes between physics, chemistry, and biology.

Models generated by VEW are one-dimensional and can be thought of as simulating a virtual *water column* extending from the ocean surface to some specified depth (See Figure 1). The cross-sectional area of the column is arbitrary as the horizontal dimensions do not feature explicitly in the model equations. The column can be anchored at a specified location or can be made to drift with ocean currents, using predetermined ocean circulation data. Lateral fluxes through the side wall of the column are ignored; for the plankton, the vertical dimension is by far the most important as many of the biological processes are dictated by light (e.g. photosynthesis, hunting, predator evasion).

The column is divided into layers (typically 500 layers each 1m deep). Associated with each layer is the concentration of each of a number of user-defined chemicals and the value of various physics variables (see below). The lower boundary is open, so a constant *rain* of particulate matter (agents) falls through the base of the column; in nature, this is the key process by which atmospheric carbon is fixed and recycled to the ocean floor.

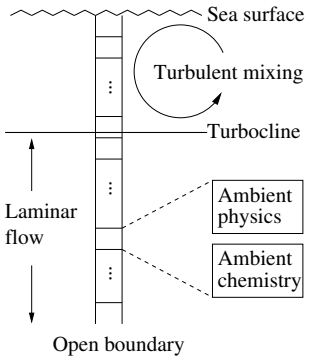

Fig. 1. One-dimensional Water Column

The combined effect of solar heating and wind creates a turbulent mixing layer in the upper section of the water column. The *turbocline* defines the boundary between the mixing layer and a region of *laminar flow* that extends to the base of the column. Agents below the turbocline that cannot swim sink under the influence of gravity; those above the turbocline are advected by the turbulence. The time-steps used in VEW simulations are sufficiently long (minimum 30 minutes) that the advection by turbulence can be well approximated by random displacement.

Because the number of individual plankton in an equivalent water column in nature can be extraordinarily large, the agents in a VEW model actually represent sub-populations of identical individuals, each with the same internal state and life history – this is the essence of the *Lagrangian Ensemble (LE)* method [9] embodied by the VEW. There is no agent-agent interaction (an $O(n^2)$ process). Instead, the LE method computes a concentration for each agent type in each layer; explicit interactions between agents are then approximated by interactions between agents and fields (an $O(n)$ process). The various biological processes are defined by the user via *primitive equations*, expressed using a mathematical modeling language called *Planktonica* [11], that is a part of the VEW. These equations are based on reproducible laboratory experiments. They include equations for motion (through turbulence/sinking), nutrient uptake, photosynthesis, respiration, reproduction and death.

Models generated in VEW are all based on the same algorithm, which is essentially a time-step simulation of the interaction between biological agents and their environment. The basic structure of the algorithm is as follows:

```
for numSteps {
    readPhysics();
    mixChemistry();
    updateAgents();
    updateChemistry();
    manageParticles();
}
```

The first line reads the pre-computed physics variables. The second mixes the chemicals in the water column that are above the turbocline depth. This is modeled by calculating the average of each chemical value for each layer in the water column that are above the turbocline. The third updates the state of the agents by applying the primitive equations to each agent separately. On typical single processor platforms, this task accounts for a substantial proportion of the overall execution time (typically over 90%). The fourth line performs chemical budgeting. Specifically, it updates the chemistry levels in the water column to reflect the usage and/or production of those chemicals by the agents. The final line invokes a particle manager which splits and/or combines agents in order to keep the total number of agents within defined bounds – this is essentially a variance reduction technique that seeks to balance statistical accuracy (variance) and execution time.

III. Parameterization

Virtual ecology models can be implemented using a number of different platforms including single or multi-processor systems running pure software implementations; custom hardware such as FPGAs, application specific integrated circuits (ASICs), or graphics processing units (GPUs); or hybrid systems that combine software and custom hardware. For any implementation, the performance depends on the complexity of the ecosystem being modeled, the underlying acceleration

platform, and the efficiency of the actual implementation (how the model is mapped to the platform). This section identifies a set of *model parameters*, *platform parameters*, and *mapping scenarios* that determine the performance and system requirements of an implementation.

A. Model Parameterization

It is important to understand the main loop of the algorithm and the data dependencies between computational tasks in order to identify the model parameters which affect performance. Figure 2 illustrates the main loop of the algorithm and the corresponding data dependencies. Note that the *mix chemistry* and *update chemistry* tasks have been combined for simplicity.

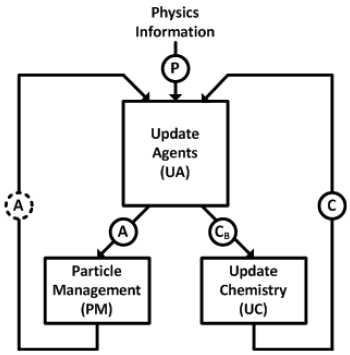

Fig. 2. Block diagram of the outer loop of the algorithm

The first computational task is the update agents (UA) task. This task cannot begin until all the physics information (P) and the updated chemistry information (C) is available because both information sets are accessed pseudo-randomly by the UA kernel(s). The UA task, however, does not have to wait for all the updated agent information (A) to arrive because the agent information is accessed sequentially. Therefore, the UA task can process the agent information as it arrives and send it back (if necessary) as soon as it is processed. The update chemistry (UC) and particle management (PM) tasks must both wait for the UA task to finish before beginning, because the UA task changes the chemistry budgeting information (C_B) and agent state (A) information pseudo-randomly.

Table I summarizes the model parameters and gives example values for the implementation described in Section V. The *T* parameter represents the number of time steps that will be simulated. Simulation periods can range from a few months to many years and a typical time step size is half an hour, therefore *T* typically ranges between 10^3 and 10^5. The *N* parameter represents the number of agents that are simulated. The I_P, I_A, I_C, and I_{C_B} parameters represent the information size (in bytes) of the physics, agent, chemistry, and chemistry budgeting information. The C_{UA-CPU}, C_{UC-CPU}, and C_{PM-CPU} parameters represent the number of CPU clock cycles needed to perform the UA, UC, and PM tasks in software, respectively. Similarly, the $C_{UA-FPGA}$, $C_{UC-FPGA}$, and $C_{PM-FPGA}$ parameters represent the number of FPGA clock cycles needed to perform the UA, UC, and PM tasks on the FPGA. The S_{UA}, S_{UC}, and S_{PM} parameters represent the size (in logic gates) of the UA, UC, and PM components when implemented on the FPGA. Finally, the L_{UA} parameter represents the latency of the UA component on the FPGA.

TABLE I
MODEL PARAMETER DESCRIPTION AND EXAMPLE VALUES

Description	Parameter	Example Values
Duration (time steps)	T	35040
Number of Agents	N	4-8k
Info Size (bytes)	I_P	6k (per time step)
	I_A	44 (per agent)
	I_C	12k
	I_{C_B}	12k
Exec Time (CPU clk cycles)	C_{UA-CPU}	2k (per agent)
	C_{UC-CPU}	49k
	C_{PM-CPU}	100k
Exec Time (FPGA clk cycles)	$C_{UA-FPGA}$	1 (per agent)
	$C_{UC-FPGA}$	1.5k
	$C_{PM-FPGA}$	23k
Area (FPGA gates)	S_{UA}	95k
	S_{UC}	11k
	S_{PM}	30k
Latency (FPGA clk cyles)	L_{UA}	156

B. Platform Parameterization

The platform implementing the ecology model affects the speed of both the computational tasks and communication of information between devices. Although the model can be implemented on a number of different platforms, this paper focuses on platforms consisting of one single core processor and one FPGA device because acceleration platforms of this type have been used previous studies and are readily available [12].

Another simplifying assumption made in this paper is that the FPGA device has enough on-chip memory to store the agent, chemistry, and physics information (for one iteration). In most cases, the agent information I_A requires the greatest amount of storage; if the FPGA did not have enough on-chip memory, the information would have to be stored off-chip and the transfer of this information could severely affect system performance (if the transfer was slower than the processing). This assumption was satisfied in the example implementation described in Section V. Furthermore, the amount of on-chip memory available on FPGAs is generally increasing with each new generation.

Table II summarizes the platform parameters and gives example values for the implementation described in Section V. The F_{FPGA} parameter represents the clock frequency of the FPGA device. This frequency depends on the implementation and the underlying technology of the device. The S_{FPGA} parameter represents the size of the FPGA device (in logic gates). Finally, the BW parameter represents the bandwidth of the communication between the host processor and the hardware device.

C. Mapping Scenarios

Although this paper focuses on a specific type of platform, there is more than one way of mapping the various tasks of the application to the platform. Table III summarizes five

TABLE II
PLATFORM PARAMETER DESCRIPTION AND EXAMPLE VALUES

Parameter	Description	Example Values
F_{FPGA}	FPGA clk freq	150 MHz
S_{FPGA}	FPGA logic capacity	160k logic gates
BW	Bandwidth between CPU and FPGA	60 Mbytes/s

different mapping scenarios of VEW generated models on platforms with one processor and one FPGA. In scenario S1, the model is implemented entirely in software using only the processor. This is the *baseline* implementation, which is used to compare against other implementations. Apart from the baseline implementation, we only consider implementation where the UA task is performed on the FPGA because this is where most (typically over 90%) of the computation is performed. Speedups achieved with the UA task performed in software would therefore be less than 10% and not worth the extra implementation effort.

In scenario S2, the UA component is implemented on the FPGA and the remaining components are implemented in software. In scenario S3, both the UA and PM components are implemented on the FPGA and the remaining components are implemented in software. In scenario S4, the UA and UC components are implemented on the FPGA. Finally, in scenario S5, the entire model is implemented on FPGA.

TABLE III
IMPLEMENTATION SCENARIOS

Task	Scenario				
	S1	S2	S3	S4	S5
UA	CPU	FPGA	FPGA	FPGA	FPGA
PM	CPU	CPU	FPGA	CPU	FPGA
UC	CPU	CPU	CPU	FPGA	FPGA

A number of factors affect which scenario is the best. As an example, there is a tradeoff between the communication overhead of transferring information between the CPU and FPGA when the UC and/or PM components are implemented in software and the area overhead of implementing the UC and/or PM components on the FPGA. If the UA kernel is small and UC and PM components are large, a greater number of the UA kernels can be implemented on the FPGA if the other components remain in software. Conversely, if the connection between the CPU and FPGA is slow, the time needed to transfer the information between the UA and the other components may become a bottleneck. These tradeoffs are examined in the following section.

IV. PERFORMANCE EVALUATION

This section describes a framework that estimates the speed of an implementation given the model parameters, platform parameters, and mapping scenario of that implementation. This framework can be used to estimate performance, identify bottlenecks, choose the best platform (if more than one is available), and choose the most efficient mapping scenario. In essence, the framework consists of a generic scheduling of tasks for each of the mapping scenarios and a corresponding expression that describes the minimum execution time of the main loop of the algorithm.

A. Scheduling

Depending on the implementation scenario, the order in which computational and communication tasks occur must be scheduled to maximize performance and satisfy data dependencies. Figure 3 shows the task schedules for Scenario S2 to S5.

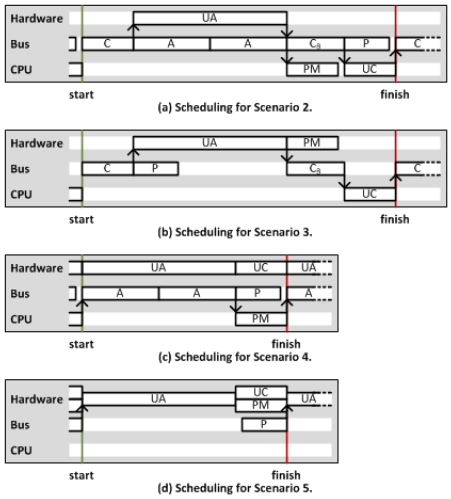

Fig. 3. Scheduling of computation and communication tasks for Scenario S2 to S5

In Scenario S1, the algorithm is implemented entirely in software and therefore the communication between tasks happens automatically as the information is stored in system memory which can be accessed by each task. The computational tasks are performed sequentially and thus the run-time is determined by simply multiplying the run-time of the UA kernel by the population size and then adding the run-time of the PM and UC tasks. Note that we are assuming a platform with a single processor, therefore only one agent can be processed at a time.

In Scenario S2, the UA kernel is implemented in hardware and thus, for each time step, the physics information (*P*), agent information (*A*), and chemistry information (*C*) must be sent from the system memory to the FPGA and then the updated agent (*A*) and chemistry budgeting information (C_B) must be sent back to the system memory. Because the physics and chemistry information are accessed randomly, P and C must be sent completely before the UA task begins, and C_B must be returned before the UC can begin. On the other hand, because the agent information is accessed in sequential order, the transmission of *B* and *D* can be synchronized with the UA task.

The scheduling of Scenario 2 is illustrated in Figure 3(a). For each iteration, *C* is sent to the hardware device; then *A* is sent, processed by UA in hardware, and then returned to system memory; then the PM is performed on the processor while C_B is returned; and finally the UC is performed and P

(for the next iteration) is sent to the hardware. The run-time of one iteration of the main loop for implementation Scenario S2 is described by the following expression.

$$T_{Loop} = T_C + Max\{T_{UA}, T_A + T_A\} + Max\{Max\{T_{C_B}, T_{PM}\} + T_{UC}, T_{C_B} + T_P\} \quad (1)$$

The expression is a summation which includes three maximum expressions, denoted *Max*. Intuitively, the first maximum term represents the time required to send, process, and return the agent information. The second and third maximum terms represent the time required to return C_B, perform PM and UC, and return A.

In Scenario S3, both UA and PM are implemented in hardware which eliminates the necessity of sending the agent information to and from hardware. This is significant because in many cases the agent information accounts for most of the information and, depending on the platform, transferring this information between system memory and the hardware device can take longer than processing it. Figure 3(b) illustrates the scheduling for Scenario S3. The corresponding run-time is shown in the bottom row of the third column in Table IV.

In Scenario S4, the UA and UC are implemented in hardware and the PM is performed in software. This eliminates the necessity of sending the environment information to and from hardware. It is also convenient because the PM, UC, and sending of the physics information (for the next iteration) can all be performed concurrently.

Finally, in Scenario 5, everything is implemented in hardware which eliminates all communication apart from sending the physics information from the system-memory to the hardware device. Also, assuming enough hardware resources are available to implement the UC and PM tasks, can be performed concurrently.

B. Performance Evaluation

Given the set model parameters and platform parameters, the overall speed of different implementation scenarios can be determined using framework described in Table IV. Rows 1–8 give a breakdown of the time required to perform individual computional and communication tasks for each implementation scenario described in III-C. Row 9 gives the time required to perform one iteration of the algorithm given the scheduling assumptions made in IV-A.

Rows 1–5 give the time needed to send information between the CPU and the FPGA for each mapping scenario. In each case, the time is either the amount of information divided by the bandwidth of the connection between the CPU and the FPGA or blank because the communication task is not necessary for that scenario. Rows 6–8 give the time needed to process the UA, PM, and UC tasks, respectively. The time needed for the UA task depends on a number of parameters. When implemented on the CPU, each agent is updated sequentially and therefore the time required is the number of agents (N) multiplied by the number CPU clock cycles per agent (C_{UA-CPU}) divided by the clock frequency of the CPU (F_{CPU}). When implemented on the FPGA, agents can be updated in parallel (if the size of the FPGA is big enough for multiple UA kernels). The time required is therefore N divided by the number of kernels that can fit on the remaining FPGA resources plus the latency of the kernel divided by the clock frequency of the FPGA. The number of UA kernels that can fit on the FPGA depends of FPGA size (S_{FPGA}), the UA kernel size ($S_{UA-FPGA}$), and the size of the PM and UC components if they are also implemented on the FPGA. Finally, the time needed for the PM and UC task is roughly the number of operations (for the CPU) or the number of clock cycles (for the FPGA) divided by the clock frequency of that device.

Estimating the performance of an implementation is straight-forward. After obtaining the values of the *model* and *platform parameters* and determining which *implementation scenario* to use, the performance is determined by evaluating the corresponding expression in Table IV. Similarly, choosing the fastest implementation scenario involves evaluating the expression for each scenario and selecting the fastest one. Choosing the most appropriate platform for a given ecosystem model involves obtaining the model parameters, platform parameters (for each available platform), evaluating the expression for each scenario for each of the platforms, and finally selecting the fastest overall platform and scenario. An example showing how the framework can be used is presented in the following section.

V. Example

This section describes the implementation of a typical VEW generated ecological model on an FPGA-based acceleration platform. The example model is small-scale VEW generate model comprising a single agent type (a form of marine phytoplankton) whose behavior is influenced by the values of the ambient temperature, light, and nutrient fields (ammonium, nitrate and silicate) at its current depth. Although VEW models conventionally incorporate biofeedback to the physical environment (chlorophyll in the phytoplankton would normally affect turbidity and thus the passage of solar energy through the column), this has been removed for simplicity. Instead, the physical environment (a set floating point variables in the model) are pre-computed for the experiment. Despite being simplified, the model retains the basic structure of large-scale models.

A. Description of Hardware Implementation

The way the algorithm is mapped to the FPGA hardware has a significant effect on the performance and resource requirements of the final implementation. For this example, the emphasis was placed on performance. The implementation exploits parallelism at three levels.

First, *coarse-grained* parallelism is exploited by efficiently scheduling the high-level computational tasks of the algorithm as described in Section IV-A.

Second, *task-level* parallelism is exploited by pipelining each of the computational tasks. Pipelining divides operations

TABLE IV
PERFORMANCE EVALUATION FRAMEWORK

Task	Scenario				
	S1	S2	S3	S4	S5
P to UA	-	I_P/BW	I_P/BW	I_P/BW	I_P/BW
A to UA	-	I_A/BW	-	I_A/BW	-
C to UA	-	I_C/BW	I_C/BW	-	-
A to PM	-	I_A/BW	-	I_A/BW	-
C_B to UC	-	I_{C_B}/BW	I_{C_B}/BW	-	-
UA	$N \cdot \frac{C_{UA-CPU}}{F_{CPU}}$	$\frac{\frac{N}{\lfloor \frac{S_{FPGA}}{S_{UA}} \rfloor} + L_{UA}}{F_{FPGA}}$	$\frac{\frac{N}{\lfloor \frac{S_{FPGA}-S_{PM}}{S_{UA}} \rfloor} + L_{UA}}{F_{FPGA}}$	$\frac{\frac{N}{\lfloor \frac{S_{FPGA}-S_{UC}}{S_{UA}} \rfloor} + L_{UA}}{F_{FPGA}}$	$\frac{\frac{N}{\lfloor \frac{S_{FPGA}-S_{PM}-S_{UC}}{S_{UA}} \rfloor} + L_{UA}}{F_{FPGA}}$
PM	$\frac{C_{PM-CPU}}{F_{CPU}}$	$\frac{C_{PM-CPU}}{F_{CPU}}$	$\frac{C_{PM-FPGA}}{F_{FPGA}}$	$\frac{C_{PM-CPU}}{F_{CPU}}$	$\frac{C_{PM-FPGA}}{F_{FPGA}}$
UC	O_{UC}/F_{CPU}	$\frac{C_{UC_FPGA}}{F_{CPU}}$	$\frac{C_{UC-FPGA}}{F_{CPU}}$	$\frac{C_{UC-FPGA}}{F_{FPGA}}$	$\frac{C_{UC-FPGA}}{F_{FPGA}}$
T_{Loop}	$T_{UA} + T_{PM} + T_{UC}$	$T_C + Max\{2*T_A, T_{UA}\} + Max\{Max\{T_{C_B}, T_{PM}\} + T_{UC}, T_{C_B} + T_P\}$	$T_C + Max\{T_{UA}, T_P\} + Max\{T_{PM}, T_{C_B} + T_{UC}\}$	$Max\{2 * T_A, T_{UA}\} + Max\{T_{UC}, T_P, T_{PM}\}$	$T_{UA} + Max\{T_{PM}, T_{UC}, T_P\}$

into smaller steps which decreases the critical-path delay but introduces latency. Pipelining is effective for operations performed for many iterations because the latency overhead becomes negligible compared to the overall time required to perform every iteration. As an example, an unpipelined 32-bit floating-float multiplier implemented on a Virtex-4 FPGA has a critical-path delay of 25ns and takes one clock cycle. The same multiplier with pipelining has critical-path delay of 5.6ns and a latency of 5 clock cycles. A computation that performs *M* iterations requires *M**25ns with the unpipelined multiplier and only (*M*+5)*5.6ns with the pipelined adder.

Third, *data-level* parallelism is exploited by identifying which arithmetic operations within each component can be performed concurrently. This depends on the data dependencies within the components. Data-level optimizations were used to accelerate the MC, UC, UA, and PM tasks.

The MC task mixes the chemical values in the water column that are above the turbocline depth. This involves taking a sum of the chemical concentrations for each chemical type in each layer and then dividing each by the number of layers. Since each chemical value is independent, the summations can be performed concurrently. In the example model, there are three chemical types (ammonium, nitrate, and silicate) and each is summed concurrently. The divisions at the end of the calculation is performed sequentially to save area (since only one divider is required instead of three).

The UC task updates the chemical environment to reflect the usage and/or production of the chemicals by the agents in the current time-step. More specifically, the task involves a subtraction (for chemicals used) and an addition (for chemicals produced) for each chemical type and each water layer. As with the MC task, the operations for each chemical type are performed concurrently.

The UA task updates the state of the agents. In terms of performance, this is the most critical task of the application because it evaluates a large number of the primitive equations for each individual agent. In software, these operations are performed sequentially; in hardware, however, many can be performed currently. The amount of parallelism depends on the data dependencies. Figure 4 illustrates how data-level parallelism is exploited in the example implementation of the UA task.

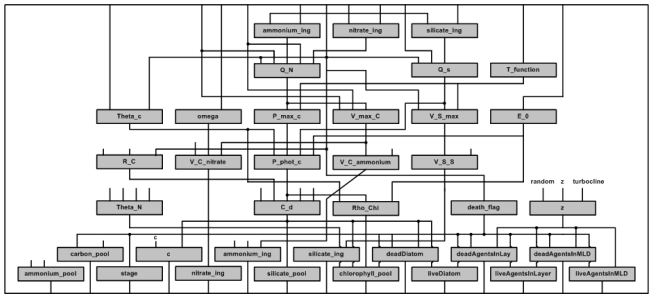

Fig. 4. Data-level parallelism in the update agent (UA) task.

The grey blocks in the figure are the primitive equations and the lines are floating-point values. Each block consists of a number of floating-point operations (to implement the equation). Not illustrated in the figure are the pipeline registers that are inserted between operators to ensure the data values remain aligned as they flow through the pipeline. The important thing to note from the figure is that horizontally adjacent blocks are computing primitive equations concurrently instead of sequential (as they would be in software).

Finally, the PM task manages the size and number of agents. Specifically, it splits agents that are too large and merges agents that are too small. To achieve this, the task requires short lists of the largest and smallest agents. In software, these lists are generated using Quicksort (an $O(nlog(n))$ process) after the UA task has updated the agents. On the FPGA, however, this type of sorting would be too slow because the clock frequency is much lower. As a solution, the sorted lists are generated in hardware using an $O(n)$ sorting technique which employs comparators for each list entry to determine (in parallel) where to insert items in a single clock cycle. This is

expensive in terms of area because it requires comparators for each list entry, however, it significantly speeds up the sorting part of the PM task which improves overall performance.

B. Tools and Libraries

The hardware version of each task of the model is implemented using the VHDL hardware description language. Higher level hardware description languages, such as SystemC and HandleC could also have been used, however, the lower level description provides more control over the final implementation which makes it easier to investigate low-level optimizations later on. VHDL is used instead of Verilog mainly because of the available VHDL floating-point libraries.

VEW generated models use double precision floating-point variables to represent all physics, chemistry, and agent state information. Although the literature suggests that future FPGAs will have hard floating-point arithmetic macro blocks, floating-point arithmetic in current FPGA must still be implemented using the programmable fine-grained resources of the FPGA. The example model employs floating-point addition, subtraction, multiplication, division, exponential (e^x), and power (x^k) operations. These operations are implemented using the publicly available VHDL floating-point library described in [13].

The hardware part of the implementation was synthesized using the Synplify Pro CAD tool [14] and the software part of the implementation, which is written in C, was compiled using gcc with optimizations turned on. Finally, the interface between the hardware and software, which uses the HyperTransport bus on the RCHTX board, was implemented using Handle-C libraries and the compiler provided by Celoxica [12].

C. Model Parameters

In order to validate the evaluation framework described in Section IV, the plankton model was implemented five ways; one implementation for each scenario presented in Table III.

The model parameter values for the example model are listed in the last column of Table I. The model simulates a 2 year period of plankton life in half hour time steps (D=35040). The simulation has an initial population (N) of 4k; however, the value varies during the simulation and increases to a maximum of approximately 8k in this example. Each agent is represented using 11 floating-point values (I_A=44 bytes). The chemistry information (I_C) and chemistry budgeting information (I_{C_B}) each require 12k bytes to store 6 different floating-point chemistry values for each of the 500 layers in the column of water. Similarly, the physics information (I_P) requires 6k bytes for the depth, temperature, and solar irradiance information for each layer. The size of the UA, UC, and PM components when implemented on the Xilinx Virtex-4 FPGA is 95k, 11k and 30k gates, respectively. The UA kernel is large because it incorporates a significant number of floating-point arithmetic operations (52 adders, 34 multipliers, and 22 dividers) and the PM component is large because it uses the expensive insertion sort technique. In terms of FPGA cycle count, the UA kernel is fully pipelined and therefore requires only one clock cycle per agent. This pipelining, however, gives the UA kernel a latency of 156 clock cycles. The UC and PM components require 1.5k and 23k FPGA clock cycles, respectively. For the PM, this is an average but it depends on the number of agents, which varies throughout the simulation. Finally, the UA, UC, and PM tasks require approximately 2k, 49k, and 100k CPU clock cycles respectively. These values were estimated by averaging the run-time of each task over the entire simulation (in software) and dividing by the clock frequency of the CPU.

D. Platform Parameters

The platform consists of a HP Proliant DL145 G3 Server with a 1GHz AMD Opteron 2200 series CPU and a Celoxica RCHTX-XV4 acceleration board. The acceleration board features a Virtex-4 FPGA with 152k logic cells and 288 18kb block RAMs which connects to host processor through a HyperTransport bus with a bandwidth of 60M bytes/s. The maximum FPGA clock frequency for the example implementation (determined by the synthesis tool) is 150 MHz. These platform parameters values are summarized in the last column of Table II.

E. Performance

Table V evaluates the performance of the example implementation using the framework described in Section IV and the example model and platform parameters. Rows 1–8 give the time required for each task. Row 9 gives the estimated time required to execute one iteration of the simulation and Row 10 gives the corresponding speedup relative to the software implementation. Note that these estimates are based on functional simulations and clock frequencies obtained from the place and route tool. We expect the speedups of the final implementation to match these simulated speedups nearly exactly because we account for every clock cycle except for a very small number of handshaking cycles between host CPU and the FPGA.

TABLE V
EXAMPLE EVALUATION

Task	Scenario				
	1	2	3	4	5
P to UA	-	1k	1k	1k	1k
A to UA	-	44k	-	44k	-
C to UA	-	1k	1k	-	-
A to PM	-	44k	-	44k	-
C_B to UC	-	512	512	-	-
UA	75k	410	410	410	410
PM	625	625	1533	625	1533
UC	306	306	306	100	100
T_{Loop} (ns)	7.6k	9.1k	3.5k	8.9k	1.9k
Speedup	1.0	0.8	21.3	0.9	39.1

A number of useful results are reported in the table. First, the table shows that mapping scenario S5 produces greatest speedup, which is 39.1 times faster than the software implementation. In this scenario, everything is implemented on the FPGA and only the physics information needs to be sent to the FPGA for each iteration (time step). Second, the table shows that mapping scenario S2 and S4 actually reduces the

performance of the model slightly compared to the software implementation. This slowdown occurs because of the time it takes to send the agent information to and from the FPGA is greater than the time it takes of update the agents. Scenario S3 provides the second greatest speedup (21.3 times faster) because it avoids sending the agent information to and from the FPGA because both tasks that use the agent information (UA and PM) are implemented on the FPGA. Scenario S3 is slower than Scenario S5 due to time needed to send and process the chemistry information.

In general, the results suggest that significant speedups can be achieved as long as the communication bottleneck that occurs when communication of the agent information takes longer than the agent update processing can be avoided. The most affective way of avoiding this bottleneck is to implement the particle management (PM) component on the FPGA so as to eliminate the need to transfer the agent information altogether. Another approach is to reduce the amount of agent information by reducing P and/or $I_{B/D}$. One way of reducing $I_{B/D}$ is to use the technique described in [15] which minimizes the wordlengths of data variables based on the amount of precisions that is required. The communication bottleneck could also be alleviated by using a platform with a higher bandwidth between the CPU and the FPGA.

Using the evaluation framework to extrapolate from the example results shows that further speedups could be achieved using a larger FPGA. Using the model parameters for the example implementation and the platform parameters for a Virtex-5 FPGA with 330K logic gates produces an estimated speedup of 69X. Some of the speedup is provided by the faster clock frequency (180 Mhz) that can be achieved using the newer FPGA, however, most of the added performance is provided by extra logic resources which allows three UA kernels to be implemented instead of only one.

VI. Acknowledgment

The authors would like to thank Wes Hinsley from preparing the software version of the example plankton model and the Natural Science and Engineering Research Council of Canada (NSERC) for their financial support of this research.

VII. Conclusions and Future Work

Agent-based modelling involves updating the state of each agent in each time step of a simulation and this yields significant amounts of parallelism. By exploiting this parallelism at various levels, an example model implemented on a commercial FPGA-based acceleration platform achieved a speedup of 39.1X at 150 MHz compared to a software implementation at 1.0 GHz.

The other main contribution of this paper is a framework that can be used to determine potential speedup for specific ecosystem models and acceleration platforms. The framework is also useful for understanding where performance can be gained or lost. In this paper, the framework was used to determine the best way to implement an example plankton model on an RCHTX acceleration platform so as to avoid communication bottlenecks. Further analysis using the framework also shows that greater speedups would be possible using either larger or multiple FPGA(s).

This paper proposes an infrastructure that allows oceanographers to automatically implement their models on FPGA-based acceleration platforms. The next step is to develop a CAD tool that analyzes results from the performance evaluation framework and then generates the appropriate HDL code, similar to that of the example implementation described in this paper, which could be synthesized and finally implemented on the acceleration platform.

Other future work includes a comparison of ecological modeling on other platform types. Specifically, the platforms with GPUs are promising because, like graphics, ecological modeling involves a large amount of floating point operations on mostly streaming data. Similarly platforms with multiple CPUs, FPGAs, or GPUs might also promising. As an example, multi-CPU platforms could be used to update multiple agents in parallel and would not require any special hardware to be designed.

References

[1] R. C. Cheung, N. J. baptiste Telle, W. Luk, and P. Y. Cheung, "Customizable elliptic curve cryptosystems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2005.

[2] M. deLorimier and A. DeHon, "Floating-point sparse matrix-vector multiply for fpgas," in *Proceedings of the Intl. Symp. on Field-Programmable Gate Arrays (FPGA)*, 2005, pp. 75–85.

[3] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson, "A reconfigurable, power-efficient adaptive viterbi decoder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 4, pp. 484–488, April 2005.

[4] D. Thomas, J. Bower, and W. Luk, "Automatic generation and optimization of reconfigurable financial monte-carlo simulations," in *Proc. of the IEEE Intl. Conf. on Application-Specific Systems, Architectures and Processors*, 2007, pp. 168–173.

[5] W. Hinsley, A. Field, and J. Woods, *Computational Science*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, July 2007, vol. 4487, ch. Creating individual based models of the plankton ecosystem, pp. 111–118.

[6] E. Nogueira, J. Woods, C. Harris, A. Field, and S. Talbot, "Phytoplankton co-existence: Results of an individual-based simulation model," *Ecological Modelling*, vol. 198, no. 1-2, pp. 1–22, September 2006.

[7] C.-C. Liu and J. Woods, "Prediction of ocean colour: Monte carlo simulation applied to a virtual ecosystem based on the lagrangian ensemble method," *International Journal of Remote Sensing*, vol. 25, no. 5, pp. 921–936, 2004.

[8] M. Sinerchia, "Testing theories on fisheries recruitment," PhD Dissertation, Imperial College London, Department of Earth Science and Engineering, 2007.

[9] J. Woods, "The lagrangian ensemble metamodel for simulating plankton ecosystems," *Progress in Oceanography*, vol. 67, no. 1-2, pp. 84–159, 2005.

[10] J. Woods, A. Perilli, and W. Barkmann, "Stability and predictability of a virtual plankton ecosystem created by an individual-based model," *Progress in Oceanography*, vol. 67, no. 1-2, pp. 43–83, 2005.

[11] W. Hinsley, "Planktonica: A system for doing biological oceanography by computer," PhD Dissertation, Imperial College London, Department of Computing, 2005.

[12] Celoxica, *Celoxica RCHTX-X4V Datasheet*, Celoxica Ltd., 2006.

[13] J. Detrey and F. de Dinechin, "Fplibrary v0.91 user documentation," in *www.ens-lyon.fr/LIP/Arenaire/Ware/FPLibrary/*. Ecole Normale Superieure de Lyon, November 2008, pp. 1–16.

[14] Synplicity, *Synplify Pro Datasheet*, Synplicity, 2008.

[15] D. Lee, A. Gaffar, R. Cheung, O. Mencer, W. Luk, and G. Constantinides, "Accuracy-guaranteed bit-width optimization," *IEEE Transactions on Computer Aided Design*, vol. 25, no. 10, pp. 1990–2000, 2006.