Hybrid Interconnect Design for Heterogeneous Hardware Accelerators

Cuong Pham-Quoc ⋆, Jan Heisswolf ⋄, Stephan Werner ⋄, Zaid Al-Ars ⋆, Jürgen Becker ⋄, Koen Bertels ⋆
⋆ Delft University of Technology, Netherlands, {P.PhamQuocCuong, Z.Al-Ars, K.L.M.Bertels}@tudelft.nl
⋄ Karlsruhe Institute of Technology, Germany, {heisswolf, stephan.werner, becker}@kit.edu

***Abstract*—The communication infrastructure is one of the important components of a multicore system along with the computing cores and memories. A good interconnect design plays a key role in improving the performance of such systems. In this paper, we introduce a hybrid communication infrastructure using both the standard bus and our area-efficient and delay-optimized network on chip for heterogeneous multicore systems, especially hardware accelerator systems. An adaptive data communication-based mapping for reconfigurable hardware accelerators is proposed to obtain a low overhead and latency interconnect. Experimental results show that the proposed communication infrastructure and the adaptive data communication-based mapping achieves a speed-up of 2.4× with respect to a similar system using only a bus as interconnect. Moreover, our proposed system achieves a reduction of energy consumption of 56% compared to the original system.**

I. Introduction

The rapid progress in technology enables to integrate more and more transistors on a single chip. Homogeneous multi- and many-core architectures and heterogeneous Systems-on-Chip (SoC) were introduced to utilize such large numbers of transistors efficiently.

In the past years, a trend towards heterogeneous on chip platforms can be observed. Intel's Atom E6x5C Processor [1] uses multiple RISC cores in combination with an FPGA fabric provided by Altera. Modern mobile devices are also based on heterogeneous SoCs combining CPUs, GPUs and specialized accelerators on a single chip.

Bus based interconnects are very area efficient for systems with a small number of cores. Once the number of cores rises, bus communication becomes inefficient due to the arbitration between all bus participants. Networks on Chips (NoC) [2] have been proposed as efficient communication infrastructure in large systems. NoCs allow parallel communication, increasing the scalability compared to buses.

In this work, we introduce a hybrid communication infrastructure containing a standard bus and a light weight NoC. The communication infrastructure can be used to accelerate heterogeneous bus-based architectures. The proposed NoC was designed with focus on low delay and low implementation costs, reducing the drawbacks of network interconnects. A communication aware mapping of the hardware accelerators to the proposed communication infrastructure is introduced. The Molen architecture extended with the NoC is used for our experiments.

The rest of this paper is organized as follows: Section II gives an overview on heterogeneous systems. Section III describes our general concept. Section IV explains the used communication infrastructure. Section V introduces the adaptive communication aware mapping of accelerator functions. Experimental results are presented in Section VI. A conclusion is given in Section VII.

978-3-9815370-0-0/DATE13/© 2013 EDAA

II. Related Work

In this section, an overview on existing heterogeneous multicore architectures, addressed by the proposed interconnect concept, is introduced. The Molen architecture [3] is a heterogeneous, shared memory system for software/hardware co-design. It combines one General Purpose Processor (GPP) with one or more dynamically reconfigurable Custom Computing Unit(s) (CCUs), so that the GPP can use them as a reconfigurable coprocessor. Each CCU has its own local memory which stores the data to be processed by the CCU. Data can be exchanged directly between the GPP and a CCU by using exchange registers through an on-chip standard bus. Alternatively the CCU can exchange data with the external memory using a centralized DMA controller. So the limitation of this architecture is that all CCUs and the GPP share one bus for accessing the external memory.

The MORPHEUS architecture [4] has an ARM9 embedded RISC processor taking care for the control flow and synchronization, and three heterogeneous reconfigurable engines (HREs) for accelerating application kernels. The control infrastructure is done via an AMBA AHB bus which connects HREs and the ARM9 processor. The control flow is also performed via exchange registers. An exotic Spidergon NoC is used to transfer data among HREs, main memory and off-chip memory. The data transfers via the NoC may be triggered by a Direct Network Access (DNA) hardware module.

A Warp processor [5] consists of a GPP, an on-chip profiler, an on-chip computer aid design module (CAD) and a warp-oriented FPGA (w-FPGA). The GPP executes the software part of an application while the critical software regions are synthesized and mapped onto the w-FPGA. The selection, synthesis and mapping are done automatically by the profiler and the CAD module. The w-FPGA and the processor share the main data cache by using a mutually exclusive execution model. The main process, CAD module and the w-FPGA are connected together through an on-chip standard bus.

LegUp [6] is an open source high-level synthesis tool for FPGA-based processor/accelerators systems. The target system contains a processor connecting with custom hardware accelerators through a standard on-chip bus interface. In the current version a shared memory architecture is used for exchanging variables between the processor and the accelerators. The shared memory uses an on-FPGA data cache and off-chip memory.

III. Overview and Concept

Bus and NoC are two interconnect architectures for System on Chip. Compared to the NoC architecture, the bus architecture has

TABLE I
COMPARISON OF A *lite*NOC ROUTER, A HERMES ROUTER AND A LIPAR ROUTER

PARAMETER	HERMES	*lite*NoC	*lite*NoC Freq.	LiPaR	*lite*NoC	*lite*NoC Freq.
Latency (1Flit,1Hop) / (4Flit,2Hop)	12 / 28 cycles	3 / 8 cycles	3 / 8 cycles	10 / 28 cycles	3 / 8 cycles	3 / 8 cycles
Used FPGA Device	Xilinx XC2V1000			Xilinx XC2VP30		
Clock Freq. (*MHz*)	25	80	135	33.33	65	115
Throughput (*Mbits/s*)	500	3200	5400	1333	2600	4600
Slices / LUTs	278 / 555	275 / 459	330 / 547	352 / 772	583 / 957	706 / 1283
Flip Flops / BRAMs	172 / 0	134 / 0	197 / 0	478 / 10	240 / 0	358 / 0

some certain advantages such as being directly compatible with most available IPs including GPP [7]. However, the competition of the connected modules to access the bus introduces arbitrary latencies. Meanwhile, the NoC architecture is emerging as a high level interconnect solution ensuring parallelism and high performance while issues such as latency and high area cost need to be addressed.

Most heterogeneous architectures introduced in Section II use only a bus as interconnect (as depicted in Figure 1(a)). In this work, we use both the NoC and the bus as interconnect between computing cores to make use of the advantages both types provide. The NoC is used to transfer data from one hardware accelerator to another while the bus is used to exchange control parameters as well as data between the GPP and the hardware accelerators. Figure 1(b) depicts the concept system. In this architecture, we assume each hardware accelerator has its own local memory to improve the performance. Using only the NoC as interconnect is an alternative solution. However, this solution will be incurred a higher hardware overhead for the network interface at the GPP and higher delay in the communication between the GPP and the local memory compared to the bus.

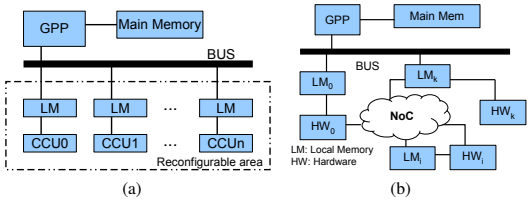

Fig. 1. (a) The Molen architecture; (b) Overview of the proposed hybrid design

There are two ways to map the connections: static and adaptive mapping. In the simple static mapping strategy, all hardware accelerators are connected to the NoC routers. All the local memories are also connected to the bus to communicate with the GPP. However, this way of connecting will cause high area overhead due to more routers are required. In the literature, most mapping research focuses on mapping heterogeneous computing cores to a NoC only [8]. We introduce a new adaptive data communication-based mapping to map the hardware accelerators to our hybrid communication infrastructure. Another difference of our approach compared to other work is that we take the actual data transferred between cores into account instead of using static information such as task graphs.

IV. COMMUNICATIONS INFRASTRUCTURE

The proposed hybrid interconnect consists of a low overhead NoC and a on chip bus. A NoC typically comprises routers, which are connected to each other, as well as network interfaces (NIs) which represent interface components attached to the NoC. In the following sections our area-efficient, delay-optimized NoC is called *lite*NoC and its infrastructure is explained.

A. Router

*lite*NoC is a packet switching meshed NoC using wormhole flow control [9] and XY-routing. Packets are divided into a header, body and a tail flit of equal size. The buffers are located at the input ports. A header flit arriving at the input port triggers a routing and reservation of the output port. After reservation is finished, the flits of the packet can be transmitted, if a buffer slot is available in the neighboring router. A credit based flow control mechanism is implemented for buffer management.

The *lite*NoC router is optimized to decrease transmission delay, reducing the disadvantage of NoCs compared to buses. When a header flit arrives at the input buffer, the reservation of the output port can be performed in only one cycle. Once the reservation is performed, a flit available in the buffer is forwarded in one cycle to the buffer of the next router. Due to the used credit based flow control, no wait cycles are required between transmission of flits of the same packet. The minimum latency in clock cycles to transfer a packet is calculated by:

$$latency = (2 * H) + S \quad (1)$$

where H is the number of hops and S the packet size.

Since we claimed to have a light weight NoC FPGA implementation with low delay, we compared the performance and resource utilization of *lite*NoC to two other packet switching router designs with XY-routing which claim to be light weight for FPGA implementation: HERMES [10] and LiPaR [11]. For a fair comparison we synthesized different versions of the NoCs using the FPGA devices and parameters of the results presented for HERMES and LiPaR in [10] and [11].

Table I shows the comparison to the HERMES NoC. A 2x2 meshed NoC with flit size of 8bit and a buffer size of 8 slots was used. For a moderate clock frequency of 80MHz *lite*NoC outperforms HERMES in all aspects. Once the clock frequency is tuned to 135MHz, *lite*NoC resource utilization is slightly higher compared to HERMES.

LiPaR is another NoC which is optimized for an efficient FPGA implementation. In [11] synthesis results of a standalone LiPaR router are presented. These results are also summarized in Table I as well as the results of a *lite*NoC router with equal parameters (flit size 8bit, buffer size 16 slots) synthesized for the same Xilinx XC2VP30 device. For a standalone router LiPaR outnumbers the *lite*NoC router resource utilization. *lite*NoC outperforms LiPaR regarding latency as well as clock frequency an has thus a much better performance and throughput. Additional synthesis results presented in [11] show that *lite*NoC outnumbers LiPaR in terms of slice utilization for a 2x2 NoC.

B. Network interface

Depending on the communication topology between hardware accelerators of a specific application, the output of one hardware accelerator can be sent to the local memory of another hardware accelerator through the NoC or be written back to its local memory. The NI attaching the hardware accelerator to the NoC can decide by analyzing the memory addresses of incoming requests, weather data are forwarded to the local memory or transmitted to another node via the NoC. Therefore, the local memory of each accelerator is also connected to the NoC. This NI connecting the memory decodes the flits from the NoC into a data part and an address part used as input signals for the memory.

C. Bus-based interconnect

The second part of the hybrid interconnect is an on chip bus used for the communication between the GPP and the hardware

accelerators. The communication intends to send controls from the GPP to the hardware accelerators and exchange data. The GPP uses the bus to transfer data between the main memory and the local memory of the hardware accelerators. It is also used to connect peripherals using standard bus interfaces. Data exchange between the GPP and the hardware accelerators use the bus, while data exchange among the hardware accelerators is performed by the NoC.

V. Adaptive Data Communication-based Mapping

In this section, an adaptive data communication-based mapping strategy for the hybrid communication infrastructure is proposed. The communication topology among functions in the application decides whether a hardware accelerator and its local memory is connected to the bus only or the NoC only or to both. We use the QUAD [12] open source toolset to analyze the data exchange inside the application and generate the data communication graph.

There are three different cases in which the hardware accelerator receives data: 1) from other hardware accelerators only (C_1); 2) from the GPP only (C_2); and 3) from both (C_3). Similarly, there are three different cases in which the hardware accelerator sends data: 1) to other hardware accelerators only (P_1); 2) to the GPP only (P_2); and 3) to both (P_3).

There are two options for the connection between the hardware accelerator and the NoC: 1) either the hardware accelerator is not connected with the NoC (H_1) or 2) it is connected with the NoC (H_2). Similarly, there are three options for the connection of the local memory with the bus and the NoC: 1) the local memory is connected only to the bus (M_1); 2) or to the NoC (M_2); or 3) it is connected to both (M_3).

The proposed adaptive data communication-based mapping is shown in Equation 2 which decides the interconnect for a hardware accelerator and its local memory based on the communication of the hardware accelerator and other components (the GPP and other hardware accelerators).

$$f : Communication \rightarrow Interconnect \tag{2}$$

where the $Communication = \{C_1, C_2, C_3\} \times \{P_1, P_2, P_3\}$ and the $Interconnect = \{H_1, H_2\} \times \{M_1, M_2, M_3\}$.

Table II shows the mapping of the communication topology to the interconnect topology. The interconnect value $\{H_1, M_2\}$ is not feasible as the result of the hardware accelerator will be inaccessible by any other function.

TABLE II
Adaptive mapping function

Communication	Interconnect
$\{C_1, P_1\}$	$\{H_2, M_2\}$
$\{C_1, P_2\}, \{C_3, P_2\}$	$\{H_1, M_3\}$
$\{C_1, P_3\}, \{C_3, P_1\}, \{C_3, P_3\}$	$\{H_2, M_3\}$
$\{C_2, P_1\}, \{C_2, P_3\}$	$\{H_2, M_1\}$
$\{C_2, P_2\}$	$\{H_1, M_1\}$

VI. Experiments

A. Setup

A Xilinx ML510 [13] board containing a XC5VFX130T-FF1738 FPGA device is used to setup our experiments. First the experimental applications are executed on the hardwired PowerPC at 400MHz to get the software execution time. We then implement the applications with the Molen platform (depicted in Figure 1(a)) in which the hardware accelerators (executed at 100MHz) are mapped onto reconfigurable area and the PowerPC is used as the GPP. Due to the hardware limitation, at most 5 CCUs are accelerated with the Molen. Finally, the applications are implemented with the Molen platform using the proposed communication infrastructure with both the bus and the NoC (called Molen with NoC). A Xilinx PLB bus and the 3×2 *lite*NoC (run at 100MHz) are used as the hybrid communication infrastructure as shown in Figure 1(b).

Four applications are used for the experiments. Here, we present the detail of the Canny edge detection application [14]. The 4 most computationally intensive functions of the Canny application are chosen to be accelerated on the hardware accelerators. The application is analyzed by the QUAD tool to extract the data communication graph (as depicted in Figure 2). Accelerated functions have a "hw_" prefix. All other functions are executed on the GPP.

According to the graph, *hw_gaussian_smooth* consumes data from the GPP and sends data to another hardware accelerator; it is classified to $\{C_2, P_1\}$. Following Table II, the communication case $\{C_2, P_1\}$ is mapped onto the interconnect case $\{H_2, M_1\}$; i.e. the corresponding hardware accelerator for this function is connected to the NoC via the NI while its local memory is connected with the bus. The connections of the hardware accelerators of other functions are deduced in a similar way (see Figure 2).

We performed the experiment with the BRAM as the local memories of the hardware accelerators. Each BRAM has only two ports while local memories of the hardware accelerators for functions *hw_magnitude_x_y* and *hw_non_max_supp* are accessed by three different components (the bus, the NoC and the accelerator). Therefore, a simple multiplexer is used.

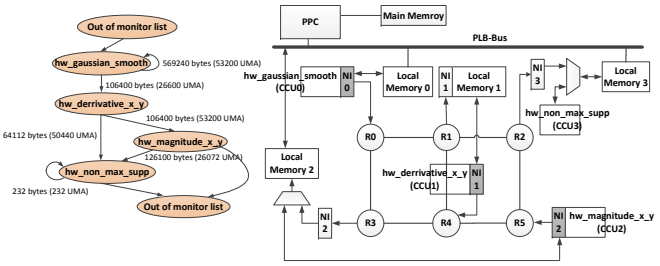

Fig. 2. The data communication graph and the Molen with NoC system for the Canny application

B. Results

Besides the Canny application, we performed the experiment with three other applications: the jpeg decoder from PowerStone benchmark [15], KLT feature tracker [16] and Fluid simulation [17]. The computation time and the communication time were measured for each hardware accelerator. In addition, the total execution time of the applications on the software, the original Molen, and the Molen with NoC were analyzed to compare the overall speed-up.

Table III shows the computation and the communication time as well as the speed-up of the Molen and Molen with NoC systems. For Canny and Fluid, the computation time of hardware accelerators (Column 8) in the Molen with NoC slightly increases compared to the original Molen (Column 4) because some hardware accelerators have to send their results to more than one local memories. However, the accompanied reduction in the communication time results in a lower overall execution time. For all applications the Molen with NoC extremely reduces the communication time by a factor of at least $1.86\times$ and up to $5.56\times$. When the proposed communication infrastructure is exploited, it increases the speed-up of the overall application and the hardware accelerators by $3.72\times$ and $6.58\times$, respectively. Figure 3(a) compares the speed-up between the two systems. This figure also shows the speed-up of hardware accelerators in the Molen system with NoC with respect to the original Molen system. Compared to the original Molen architecture, speed-ups of up to $2.4\times$ are obtained.

Table IV presents the hardware resource utilization for accelerators and communication infrastructure in the Molen and the Molen with NoC systems in terms of the number of LUTs and the number

TABLE III
EXECUTION TIME OF HARDWARE ACCELERATORS AND SPEED-UP

Application	# acc.	Software	Molen				Molen with NoC			
			comp.	comm.	acc. speedup	app. speedup	comp.	comm.	acc. speedup	app. speedup
Canny	4	41.82ms	17.72ms	5.15ms	1.83×	1.70×	17.80ms	1.47ms	2.17×	1.97×
jpeg	4	7.79ms	2.07ms	7.52ms	0.81×	0.81×	2.07ms	1.93ms	1.95×	1.86×
KLT	3	1,812ms	99.38ms	328.24ms	4.23×	2.93×	99.38ms	175.80ms	6.58×	3.72×
Fluid	5	54.60ms	24.33ms	27.62ms	1.05×	1.04×	27.55ms	4.96ms	1.68×	1.66×

comm.: communication time; comp. computation time; acc.: accelerators; app.: application

of Registers. A 3×2 *lite*NoC taking 1854 LUTs and 2122 Registers is used for all applications. Therefore we do not show these number for each application. The maximum amount of resources used for the communication infrastructure in the experiments takes only 5.8% of FPGA area.

TABLE IV
HARDWARE RESOURCE UTILIZATION (#LUTS/#REGISTERS)

Application	Original Accelerator	Molen with NoC		
		Accelerator	NI	Total
Canny	8878/12519	8999/12938	1578/1581	12431/16641
jpeg	10707/11722	10707/11722	1140/1054	13701/14898
KLT	3673/5242	3673/5242	570/527	6097/7891
Fluid	18167/28605	19848/31353	2454/2635	24156/36110

Table V gives a comparison of hardware resource utilization for the mapping using the proposed adaptive data communication-based and the static mapping in which all hardware accelerators and their local memories are connected to the NoC. The comparison takes the number of LUTs and registers used for the NIs and the multiplexers into consideration. The adaptive data communication-based mapping takes less resources than the static mapping. The proposed adaptive mapping saved up to 66.7% LUTs and registers compared to the static mapping.

TABLE V
HARDWARE RESOURCE UTILIZATION COMPARISON BETWEEN THE PROPOSED MAPPING STRATEGY AND THE STATIC MAPPING

Application	Proposed mapping (#LUTs/#registers)	Static mapping (#LUTs/#registers)
Canny	1578/1581	2280/2108
jpeg	1140/1054	2280/2108
KLT	570/527	1710/1581
Fluid	2454/2635	2850/2635

Figure 3(b) presents the comparison of the energy consumption of the Molen and the Molen with NoC systems, normalized to the consumption of the Molen system. We used Xilinx XPower Analyzer to estimate the power consumption of each application. The energy consumption is given by the product of power consumption and execution time. As shown in the figure, the Molen with NoC outperforms the Molen in all applications in terms of energy consumption. The maximum energy saved is 56% for the jpeg application.

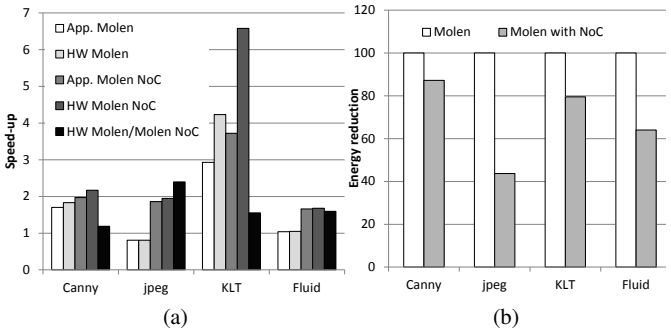

Fig. 3. (a) Overall application speed-up and hardware accelerators speed-up of Molen and Molen with NoC systems; (b) Energy consumption of Molen and Molen with NoC systems

VII. CONCLUSION

In this paper, we presented a hybrid communication infrastructure that includes a standard bus system and a newly proposed area-efficient, delay-optimized NoC for accelerating heterogeneous architectures, especially for hardware accelerator systems. The bus is used for data exchange between the GPP and the hardware accelerators while data exchange among the hardware accelerators is performed by the NoC. According to the proposed hybrid communication infrastructure, a new adaptive data communication-based mapping strategy is introduced to map the reconfigurable hardware accelerators to the communication infrastructure. We investigated the proposed approach using the Molen architecture. The original Molen was extended with the hybrid communication infrastructure. The results show that the proposed system achieves a speed-up of the overall application by 3.72× and of the hardware accelerators by 6.58× with respect to software. The speed-up of hardware accelerators with respect to the original system goes up by 2.4× while only 5.8% additional FPGA resources are required in the worst case. Due to the reduced execution time, energy reduction of up to 56% compared to the original systems, was obtained.

ACKNOWLEDGMENT

This work has been funded by the projects Smecy 100230, iFEST 100203, REFLECT 248976, the DFG TCRC "'Invasive Computing" (SFB/TR 89) and Vietnam Ministry of Education and Training.

REFERENCES

[1] "Intel® atom™ processor E6x5C series-based platform for embedded computing," 2010. [Online]. Available: http://download.intel.com/embedded/processors/prodbrief/324535.pdf
[2] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, 2002.
[3] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. Panainte, "The MOLEN polymorphic processor," *Computer*, 2004.
[4] M. Kuhnle, M. Hubner, J. Becker, A. Coppola, L. Pieralisi, R. Locatelli, G. Maruccia, T. DeMarco, F. Campi, A. Deledda, C. Mucci, and F. Ries, "An interconnect strategy for a heterogeneous, reconfigurable SoC," *Design Test of Computers*, 2008.
[5] R. Lysecky and F. Vahid, "Design and implementation of a microblaze-based warp processor," *ACM Trans. Embed. Comput. Syst.*, 2009.
[6] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "LegUp: high-level synthesis for FPGA-based processor/accelerator systems," in *FPGA*, 2011, pp. 33–36.
[7] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *DATE*, 2000.
[8] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms," *J. Syst. Archit.*, vol. 56, no. 7, pp. 242–255, Jul. 2010.
[9] L. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, 1993.
[10] F. Moraes *et al.*, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *Integr. VLSI J.*, 2004.
[11] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri, "Lipar: A light-weight parallel router for fpga-based networks-on-chip," in *GLSVLSI*, 2005.
[12] S. A. Ostadzadeh, R. J. Meeuws, C. Galuzzi, and K. Bertels, "QUAD: a memory access pattern analyser," in *ARC*, 2010.
[13] Xilinx, "Ml510 reference design," 2009.
[14] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1986.
[15] J. Scott *et al.*, "Designing the low-power M•CORE architecture," in *IEEE Power Driven Microarchitecture Workshop*, 1998.
[16] J. Shi and C. Tomasi, "Good Features to Track," in *IEEE Conference on Computer Vision and Pattern Recognition*, 1994.
[17] J. Stam, "Real-time fluid dynamics for games," in *the Game Developer Conference*, 2003.