23.10 HuMoniX: A 57.3fps 12.8TFLOPS/W Text-to-Motion Processor with Inter-Iteration Output Sparsity and Inter-Frame Joint Similarity

Jaehoon Heo, Adiwena Putra, Sungwoong Yune, Jieon Yoon, Hangyeol Lee, Jihoon Kim, Joo-Young Kim

KAIST, Daejeon, Korea

Recently, 3D human motion generation has become crucial for entertainment and media applications, such as film production and AR/VR. This process involves creating human joint movements and constructing detailed 3D meshes, like human skin, for each joint (see Fig. 23.10.1). It used to require hours or days of using motion capture suits or manually modeling each joint frame by frame. However, transformer-based diffusion models now enable the generation of joint movements within seconds from inputs such as text [1-2] or music [3]. These models start with random noise and iterate through denoising steps to produce the desired output while integrating provided inputs. Despite this advancement, two major challenges remain for real-time execution.

The first challenge is the high computational cost caused by excessive kernel iterations in diffusion models, where each iteration involves multiple transformer blocks, and the total number of iterations can even reach up to 1000. The second challenge is the long processing time of mesh construction, which takes more than a second per frame by a CPU. The widely-used skinned multi-person linear model (SMPL) [4] relies on parameters such as body pose and camera translation to construct 3D meshes. To accurately synthesize complex human skin, SMPL must be executed repeatedly per frame, progressively updating the parameters to find their optimal values. As a result, data dependencies within these repetitions, along with inter-frame dependencies, hinder GPU acceleration, making real-time operation unrealistic.

To address these challenges, we propose a novel text-to-motion processor called HuMoniX, enabling real-time human motion generation by integrating two heterogeneous engine clusters. First, a joint creation cluster (JCL) leverages the unique inter-iteration output sparsity in diffusion models, reducing computational costs by skipping redundant computations. Second, a mesh construction cluster (MCL) selectively runs SMPL to generate accurate meshes only for frames with significant movements based on inter-frame joint similarity analysis. It applies interpolation for the rest of the frames, significantly reducing overall latency. Consequently, HuMoniX achieves 57.3fps performance at 600MHz, while simultaneously running diffusion for joint creation and SMPL for mesh construction.

Figure 23.10.2 shows HuMoniX's overall architecture, comprising a top controller, JCL, MCL, and a global scratchpad (GSC). The process begins with the top controller fetching weights, random noise, and input embeddings into the JCL. Once the data is ready, the JCL, equipped with a sparsity-aware diffusion core (SDC), starts joint creation by executing denoising steps, where the SDC skips redundant computations by reusing output data stored in the GSC across different iterations. During this process, a ConMerge assistant unit (CAU) within the SDC generates control signals and minimizes the overhead of handling sparsity. After the joint data is ready, MCL initiates SMPL to construct meshes only for the necessary frames, while the JCL begins processing the next input batch. During mesh construction, MCL's three mixed-precision SMPL cores (MPCs) use floating-point (FP) data for precise construction while supporting both low- and high-precision FP to minimize the area overhead. Once the MCL completes mesh construction for the first two frames, JCL performs simple interpolation whenever it is not occupied with joint creation, effectively hiding latency. By leveraging this seamless integration of both clusters, HuMoniX efficiently manages the entire motion generation process.

Figure 23.10.3 shows how JCL reduces the computational costs in joint creation through three software-hardware co-design strategies. 1) It introduces the FFN-Reuse algorithm, which performs one dense iteration followed by N sparse iterations. In the sparse iteration, the algorithm skips redundant computations by reusing output data from dense iteration in the feed-forward network (FFN) layers, the major computational workload, up to 95%. Thus, this leverages the inter-iteration output sparsity. 2) A data-compaction mechanism called ConMerge is proposed for removing unstructured sparsity in the output matrix. First, it condenses the output matrix by removing fully sparse columns. Then, it partitions the matrix into smaller blocks that align with the hardware tiling size and merges each block up to twice, reducing columns by up to 92%. During merging, conflicts from overlapping elements are resolved by reallocating them to empty units, with traces recorded in the conflict vector. 3) A sparse-dense unified engine (SDUE), consisting of an 18×16 integer dot-product unit (INT_DPU) array, each with multipliers, a Wallace tree adder, and an accumulator, is designed for efficient handling of both sparse and dense matrix-matrix multiplication (MMUL). During sparse MMUL, it maps ConMerge results to its array while still utilizing the broadcasted data-feeding path used in dense MMUL. To facilitate this, each INT_DPU receives two horizontally broadcasted input lines: one from each input memory (IMEM) bank and another from a bank where a conflict element was moved. For weight data, the triple-buffered weight memory (WMEM) vertically broadcasts three columns from the same WMEM bank to each column of the array. Conflict switches between the IMEM and the array, along with input and weight switches within each INT_DPU, ensure the correct input rows and weight columns are selected. Additionally, a configurable SIMD engine (CFSE) supports SIMD operations with either 2-way 16b or 1-way 32b fixed-point data, maintaining quantization accuracy. During sparse iterations, it executes the 2nd FFN layer in a row-wise MMUL manner using scalar-vector multiplication, processing the unstructured results of the 1st FFN layer generated by the SDUE. This approach leads to a 5.0× reduction in latency and a 5.5× improvement in energy efficiency in the FFN layers during sparse iterations compared to dense ones.

Figure 23.10.4 illustrates how HuMoniX constructs 3D meshes in real-time by integrating MCL and JCL. After joint creation, JCL calculates inter-frame joint similarity between subsequent frames to identify major changes in human movements. For frames with significant changes, MCL trains SMPL's key parameters (0.34KB) by repeating its forward and backward paths 110× using FP data for accurate mesh construction. For frames with a minimal difference of less than 0.7 similarity, JCL performs simple interpolation, significantly reducing the latency of mesh construction per frame by 99.6% with negligible accuracy loss. To minimize the computational overhead of FP operations, MPCs include a low-precision dot product engine using FP12 (1/5/6) precision for MMUL and a high-precision SIMD engine using FP18 (1/7/10) precision for SIMD-type operations, such as loss calculation and training optimizers. This design reduces the area and memory footprints of MPC by 33.3% and 30.4%, respectively, compared to using only FP18. Additionally, a format converting unit between both engines includes an FP precision converter to handle all subnormal cases and a matrix transposer that efficiently manages the transposition required for MMUL's backward path using two-way readable register files. Finally, HuMoniX reduces the total mesh construction latency by up to 98.5% compared to running SMPL on all frames on the CPU.

Figure 23.10.5 shows how the CAU, comprising a sparsity-level classifier (SLC), SortBuffer, and ConMerge vector generator (CVG), manages ConMerge's overhead and generates control signals. Directly merging blocks with random sparsity increases latency due to excessive conflict resolution. To avoid this, it sorts columns by sparsity levels, then merges by selecting one column from dense and others from sparse levels, reducing cycle counts by 29-35%. While naive bucket sort has a complexity of O(n+k) on average and O(n^2) in the worst case, the proposed SortBuffer consistently achieves O(n). After receiving column indices and sparsity bitmask from SDUE, SLC counts non-zero bits to determine sparsity and stores data in SRAM banks, categorized from high_dense to high_sparse. If a bucket overflows, data is redirected to the next sparse or extra bucket, managed by monitoring logic. After sorting, the CVG reads a dense and a sparse row to create a bitmask map, identifies problematic columns, resolves conflicts by moving conflicting bits to empty spots, updates control signals for SDUE switches, and repeats these steps until all conflicts are resolved. Once merging two rows finishes, it merges one more sparse row in SortBuffer, writes final outputs to CVG memory, and proceeds to the next merge. Consequently, it manages the overheads and generates all the required signals to support ConMerge while occupying only 1.2% of the JCL area.

The 12mm^2 HuMoniX chip (Fig. 23.10.7), fabricated using 14nm technology, operates at 50-600MHz with a supply voltage of 0.63 to 0.94V. Figure 23.10.6 shows its measurement results and comparison table. It demonstrates robust accuracy in final 3D mesh outputs, achieving PSNR values of 23.4 to 50.4. In single batch experiments, HuMoniX, with 3.2GB/s external memory bandwidth (BW), achieves up to 57.3fps, making it 12.4 to 650.5× faster than edge GPU systems [5], despite having a smaller BW. This performance is achieved by skipping redundant computations and weight fetches in FFN layers and reducing the number of frames requiring SMPL processing. By reducing total energy consumption by 72.3%, it achieves energy efficiency of 8.9 to 11.0TOPS/W for joint creation and 12.3 to 12.8TFLOPS/W for mesh construction, respectively. These results demonstrate that it is 1729 to 3572× more efficient than comparable GPUs and 2 to 6M× more efficient than CPU-based executions. In conclusion, HuMoniX, a human motion generation processor, achieves real-time generation speeds with high energy efficiency.

Acknowledgement:
This work was supported by the Ministry of Science and ICT (MSIT), South Korea, under the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant (No. 2022-0-01037, Development of High Performance Processing-in-Memory Technology based on DRAM) and grant (No. 2022-0-01036, Development of Ultra-Performance PIM Processor SoC with PFLOPS-Performance and GByte-Memory).

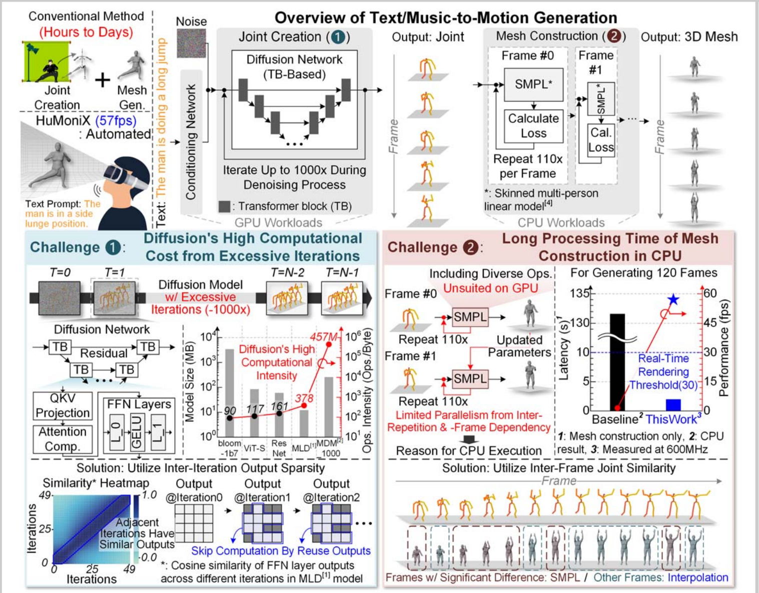


Figure 23.10.1: Overview of text/music-to-motion generation and challenges in current systems.

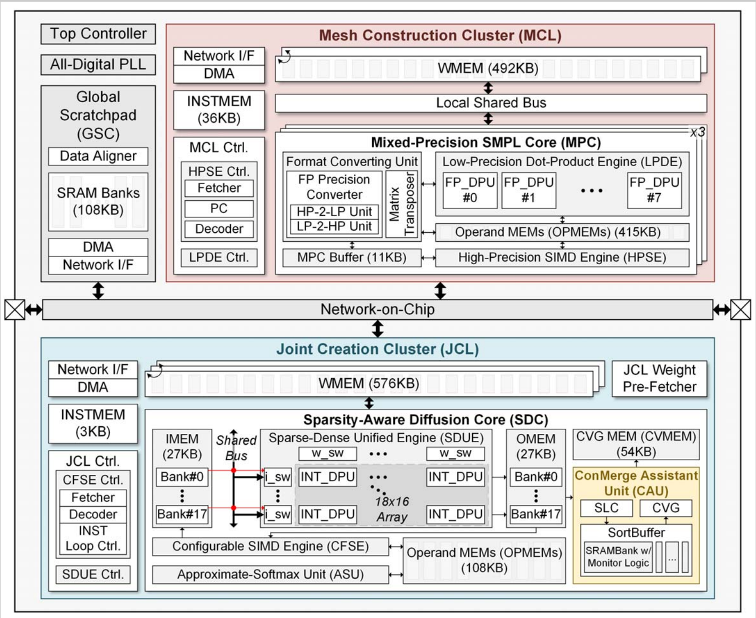


Figure 23.10.2: Overall chip architecture of HuMoniX.

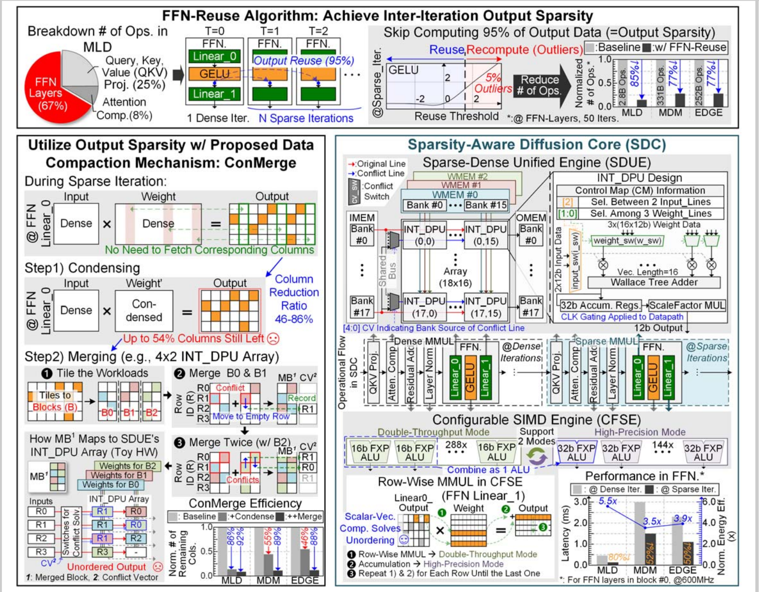


Figure 23.10.3: FFN-Reuse algorithm, ConMerge mechanism, and detailed SDC architecture.

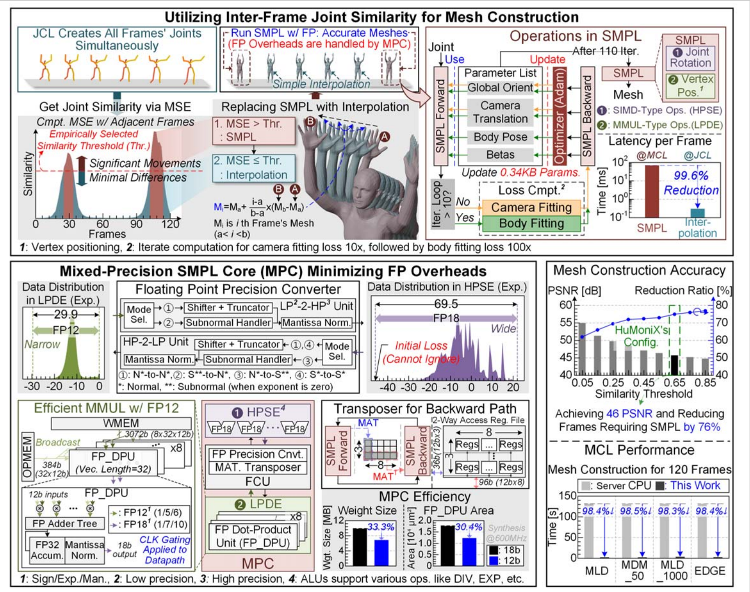


Figure 23.10.4: Real-time mesh construction system in HuMoniX and detailed MPC design.

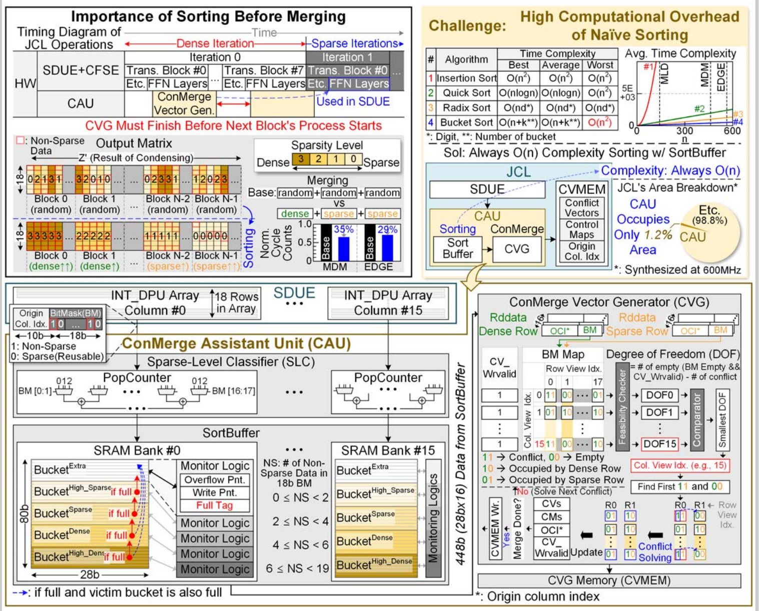
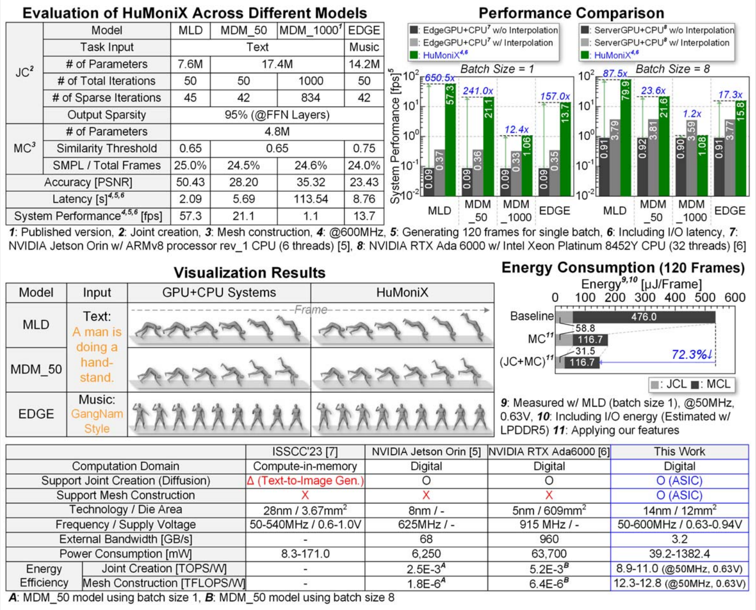


Figure 23.10.5: Importance of efficient sorting before merging and detailed CAU design.



Evaluation of HuMoniX Across Different Models

	Model	MLD	MDM_50	MDM_1000[1]	EDGE
JC[2]	Task Input	Text			Music
	# of Parameters	7.6M	17.4M		14.2M
	# of Total Iterations	50	50	1000	50
	# of Sparse Iterations	45	42	834	42
	Output Sparsity	95% (@FFN Layers)			
MC[3]	# of Parameters	4.8M			
	Similarity Threshold	0.65	0.65		0.75
	SMPL / Total Frames	25.0%	24.5%	24.6%	24.0%
Accuracy [PSNR]		50.43	28.20	35.32	23.43
Latency [s][4,5,6]		2.09	5.69	113.54	8.76
System Performance[4,5,6] [fps]		57.3	21.1	1.1	13.7

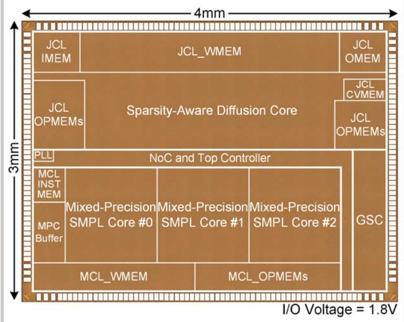
1: Published version, 2: Joint creation, 3: Mesh construction, 4: @600MHz, 5: Generating 120 frames for single batch, 6: Including I/O latency, 7: NVIDIA Jetson Orin w/ ARMv8 processor rev_1 CPU (6 threads) [5], 8: NVIDIA RTX Ada 6000 w/ Intel Xeon Platinum 8452Y CPU (32 threads) [6]

9: Measured w/ MLD (batch size 1), @50MHz, 0.63V, 10: Including I/O energy (Estimated w/ LPDDR5) 11: Applying our features

		ISSCC'23 [7]	NVIDIA Jetson Orin [5]	NVIDIA RTX Ada6000 [6]	This Work
Computation Domain		Compute-in-memory	Digital	Digital	Digital
Support Joint Creation (Diffusion)		Δ (Text-to-Image Gen.)	O	O	O (ASIC)
Support Mesh Construction		X	X	X	O (ASIC)
Technology / Die Area		28nm / 3.67mm^2	8nm / -	5nm / 609mm^2	14nm / 12mm^2
Frequency / Supply Voltage		50-540MHz / 0.6-1.0V	625MHz / -	915 MHz / -	50-600MHz / 0.63-0.94V
External Bandwidth [GB/s]		-	68	960	3.2
Power Consumption [mW]		8.3-171.0	6,250	63,700	39.2-1382.4
Energy Efficiency	Joint Creation [TOPS/W]	-	2.5E-3[A]	5.2E-3[B]	8.9-11.0 (@50MHz, 0.63V)
	Mesh Construction [TFLOPS/W]	-	1.8E-6[A]	6.4E-6[B]	12.3-12.8 (@50MHz, 0.63V)

A: MDM_50 model using batch size 1, B: MDM_50 model using batch size 8

Figure 23.10.6: Measurement results and performance comparison table.



			Chip Specifications		
Technology [nm]			14		
Die Area [mm^2]			12		
Frequency [MHz]			50-600		
Voltage [V]			0.63-0.94		
SRAM			1.87 MB		
Data Type	JCL	SDUE	INT12		
		CFSE	FXP16 / FXP32		
	MCL	LPDE	FP12		
		HPSE	FP18		
External Bandwidth [GB/s]			3.2		
Peak Power Consumption [mW]			39.18 (@50MHz, 0.63V)		
			1382.39 (@600MHz, 0.94V)		
Peak Throughput^1 [TOPS or TFLOPS]			6.71 (@600MHz)		
Performance^2					
Model Type	Joint Creation		MLD	MDM_50	EDGE
	Mesh Construction		SMPL		
Task Input			Text		Music
Latency^{3,4} [s]		Batch=1	2.09	5.69	8.76
		Batch=8	12.02	44.42	60.94
System Performance^{3,4} [fps]		Batch=1	57.3	21.1	13.7
		Batch=8	79.9	21.6	15.8
Operating Conditions			@50MHz, 0.63V		
Energy Efficiency^1	Joint Creation [TOPS/W]		11.0	8.9	9.6
	Mesh Construction [TFOPS/W]		12.8	12.3	12.8

1: 1 MAC = 2 Operations
2: For generating 120 frames
3: @600 MHz
4: Including I/O latency

Figure 23.10.7: Chip micrograph and performance summary.

References:
[1] X. Chen et al., "Executing Your Commands Via Motion Diffusion in Latent Space," *CVPR*, pp. 18000-18010, 2023.
[2] G. Tevet et al., "Human Motion Diffusion Model," arXiv:2209.14916, 2022.
[3] J. Tseng et al., "Edge: Editable Dance Generation From Music," *CVPR*, pp. 448-458, 2023.
[4] M. Loper et al., "SMPL: A Skinned Multi-Person Linear Model," *ACM Trans. on Graphics*, vol. 34, no. 6, pp. 1-16, 2015.
[5] NVIDIA Jetson Orin Nano, <https://developer.nvidia.com/embedded/learn/get-started-jetson-orin-nano-devkit>, Accessed: Sep. 2024.
[6] NVIDIA RTX 6000 Ada Gen, <https://www.nvidia.com/en-us/design-visualization/rtx-6000>, Accessed: Sep. 2024.
[7] Guo. R et al., "A 28nm 74.34 TFLOPS/W BF16 Heterogenous CIM-Based Accelerator Exploiting Denoising-Similarity for Diffusion Models," *ISSCC*, pp. 362-364, 2024.