ENGINEERING Information Technology & Electronic Engineering
https://www.academax.com/EITEE; https://www.engineering.org.cn
https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=11344065
E-mail: eng.itee@zju.edu.cn

Supplementary materials for

Zhuang CAO, Tian ZHANG, Xiaowei HE, Sheng LIU, Junzhong SHEN, 2026. Efficient mapping and flexible interconnects: accelerating 3D CNN-based lung nodule segmentation and classification on multi-FPGA. *ENGINEERING Information Technology & Electronic Engineering*, 27(6):250186.
https://doi.org/10.1631/ENG.ITEE.2025.0186

Section S1 Mathematical Justification of Fixed Sparsity Patterns

The fixed sparsity patterns observed in Fig. 4(b) can be formally derived from the structure of the Winograd transformation matrices. For $F(2^3, 3^3)$, the input transformation matrix X (size 4×4) contains 50% zero entries. When applied to a sparse input tile $\mathbf{x}$ with zero-insertion (Fig. 4(a)), the Kronecker-product nature of 3D Winograd transformation (Lavin and Gray, 2016) leads to:

$$X \otimes X \otimes X \cdot vec(\mathbf{x}) = vec((X\mathbf{x}X^T)^R X^T) \tag{S1}$$

Due to the zeros in X and the structured zeros in $\mathbf{x}$, the resulting intermediate tile m exhibits a fixed sparsity pattern independent of the actual voxel values. Specifically, for boundary tiles (requiring zero padding), 43.8% of entries are guaranteed zero; for internal tiles (no padding), sparsity reaches 87.5%. This property allows us to precompute which output positions are always zero and skip their computation entirely, which is a key insight that underpins our accelerator optimization.

Section S2 Quantification of Hardware Resource Savings

Table S1 Adder and DSP reduction in transformation templates

Template Type	[Shen et al. 2020]	Ours (Sparse-Optimized)	Reduction
TX (input transform)	32 adders	16 adders	50%
TW (filter transform)	24 adders	6 adders	75%
TM (output transform)	28 adders	16 adders	43%
EWMU (multipliers)	64 DSPs	64 DSPs	0%
Total Adders	84	38	54.8%
Total DSPs	64	64	0%

Section S3 Routing Rules and Latency Breakdown

Routing rules are defined based on packet type:

- **Multicast** (e.g., broadcasting results from dense blocks to all nodes) is always routed through the switch via routine paths, as the switch natively supports one-to-many packet duplication.
- **Unicast** (e.g., point-to-point workload transfer between specific nodes) preferentially uses fast paths when a direct link exists; otherwise, it falls back to the "routine path" via the switch.

Latency breakdown between the two path types is measured experimentally under zero-load conditions:

- **Routine path (40 GbE, TCP/IP):** Round-trip latency = $2.8\mu s$ (including switch forwarding and TCP handshake overhead).

- **Fast path (100 GbE, lightweight protocol):** Round-trip latency = $0.6\mu s$ (direct link, no handshake, minimal header processing).

This 4.7× latency reduction validates the effectiveness of fast paths for time-critical unicast operations such as workload redistribution in dense blocks.

Section S4 Overall Acceleration Framework – Stage 1–4 Details

The overall acceleration framework operates as follows:

Stage 1: Preprocessing (CPU). The host CPU retrieves raw CT data ($512 \times 512 \times 300$ voxels, $\sim$ 150 MB) from main memory and generates 3D image patches ($48 \times 96 \times 96$) through image masking, interpolation, and coordinate transformation. This produces approximately 120 patches per scan ($\sim$ 6 MB) with a latency of 45.0 ms.

Stage 2: LNS-net execution (FPGA1–4). The CPU transfers the weights of LNS-net to FPGA nodes via PCIe, adopting a model parallelism strategy whereby each FPGA node is assigned a subset of model weights. The preprocessed patches are distributed across four FPGAs (30 patches each, totaling 120 patches per scan) and loaded into off-chip memory. After computation, the CPU retrieves the resulting probability maps. The per-patch latency is 1.2 ms (transfer) + 9.2 ms (compute), resulting in a total of 1.25 seconds per full scan (120 patches × 10.4 ms).

Stage 3: Connected Component Analysis (CPU). The probability maps are thresholded and clustered on the CPU to extract candidate nodule coordinates. This reduces data size from 6 MB to $\sim$ 0.5 MB (10–20 candidates, each $16 \times 32 \times 32$) with a latency of 1.5 ms per scan.

Stage 4: LNC-net execution (FPGA5–6). The CPU transfers the candidate patches to two FPGA nodes, which host the entire LNC-net model using a data parallelism strategy. The FPGAs process batched candidates and return malignancy scores. Per-candidate latency is 0.1 ms (transfer) + 0.3 ms (compute), totaling $\sim$ 6 ms per scan for 15 candidates.

Section S5 Algorithm S1: Mapping Scheme for a Dense Block

Algorithm S1 Mapping scheme for a dense block

Input: A dense block with n layers $\langle l_1, l_2, \ldots, l_n \rangle$, FPGA Accelerators $\langle f_1, f_2, \ldots, f_p \rangle$ $(p < n)$
Output: Layer assignments for all accelerators $\langle LA_1, LA_2, \ldots, LA_p \rangle$
Notations:
  $C(j)$: Computational cost of the j^{th} layer in dense block, $C(j) = \alpha \cdot OPs(j) + \beta \cdot Dep(j)$
  $OPs(j)$: Floating-point operation count of layer l_j, $OPs(j) = 2 \times N_{in} \times N_{out} \times K^3 \times D \times H \times W$
  $Dep(j)$: Synchronization overhead of layer l_j (number of preceding layers it depends on)
  LA_i : Layer assignment set for the i^{th} FPGA accelerator
for $1 \leq i \leq p$ **do**
  $LA_i \leftarrow \emptyset$;
  $LA_i \leftarrow (LA_i \cup l_i)$;
  $Compute(f_i)$; // Start accelerator f_i with initial layer assignment
end for
$Iter = \frac{n}{p}$; // Assume n is a multiple of p for simplicity; number of iteration groups
for $1 \leq i < Iter$ **do**
  $count = 0$;
  while $(Is\_done(f_1) \vee Is\_done(f_2) \vee ... \vee Is\_done(f_p)) \wedge count \leq p$ **do**
    // Accelerator f_j completes current task
    if $Is\_done(f_j)$ **then**
      for $k > 0 \wedge k \neq j$ **do**
        $Result\_trans(f_j, f_k)$; // Transmit layer results from f_j to other accelerators
        $Compute(f_k)$; // Resume computation of f_k with new data
      end for
      $count = count + 1$;
      $LA_j \leftarrow (LA_j \cup l_{(i-1)*p+count})$ // Assign the highest $C(j)$ unassigned layer to f_j (dynamic load balancing)
    end if
  end while
  $i = i + 1$;
end for
return $\langle LA_1, LA_2, \ldots, LA_p \rangle$

Section S6 Quantitative Comparison of $p{=}f$ vs. $p{>}f$ Design Options

We quantitatively compared two design options based on three metrics: (1) **Resource utilization efficiency**—consolidating multiple accelerators on one node increases on-chip buffer contention, with BRAM utilization per accelerator dropping by 23% due to bank conflicts; (2) **Inter-accelerator communication**—when multiple accelerators reside on the same FPGA, the NI module becomes a bottleneck, increasing data transfer latency by up to 31% due to contention; (3) **Control logic overhead**—multiple accelerators per node require a centralized task scheduler consuming additional LUTs (up to 8% of total logic) and increasing critical path delay.

Based on this analysis, we adopt $\boldsymbol{p{=}f}$ as the optimal configuration, achieving 18% higher system throughput compared to a hypothetical $p{>}f$ design under the same resource constraints.

Section S7 LNS-net Mapping – Detailed Stage Descriptions

(1) Initialization. The computation begins by mapping CONV0-0 to accelerator Acc1 on FPGA node1. The results are stored in the external memory of node1 and simultaneously multicast to all other nodes, ensuring every node possesses a copy.

(2) Dense Block Computation and Workload Assignment. Execution starts with the first workload group (DB1-1 to DB1-4). FPGA node1 completes its task first and multicasts the results (while retaining a local copy in its buffers). Upon receipt, nodes 2–4 resume computation concurrently. Meanwhile, the highest-cost workload from the next group, DB1-7, is assigned to node1. This process repeats for all nodes leading up to the DB1-7 computation. Due to its significantly larger workload, DB1-7 is distributed across all nodes to minimize execution time. For instance, when node4 finishes DB1-4, node1 unicasts half of its remaining DB1-7 workload to node4. Similarly, nodes 2 and 3 receive portions of the DB1-7 workload from nodes 1 and 4 upon completing their respective tasks (DB1-6 and DB1-7). Unlike previous phases, the results from DB1-1 to DB1-6 are cached in the on-chip buffers of their respective nodes for efficient reuse by subsequent layers (e.g., results from DB1-3 are reused by DB1-5 and DB1-7 in node3).

(3) TCONV Layer Computation. The computations for TCONV2 to TCONV4 are initiated as soon as the outputs from their corresponding dense blocks become available. Each FPGA node stores the intermediate results locally, synchronizing based on the availability of results from the preceding TCONV layer.

Section S8 Scalability Analysis – Performance Model and Bottlenecks

To evaluate how our system scales beyond the current 6-FPGA configuration, we develop an analytical performance model based on the roofline model (Zhang et al., 2015) and our measured per-node throughput. The model predicts system throughput $T(N)$ for N FPGA nodes as:

$$T(N) = \min\left(N \times T_{\text{node}},\ \frac{B_{\text{switch}}}{D_{\text{comm}}},\ \frac{M_{\text{total}}}{M_{\text{node}}} \times T_{\text{node}}\right) \tag{S2}$$

where $T_{\text{node}} = 2.65$ TOPS, $B_{\text{switch}} = 640$ Gbps, $D_{\text{comm}} = 0.6$ μs/bit, and $M_{\text{total}}/M_{\text{node}}$ captures memory bandwidth sharing.

We identified bottlenecks with more nodes:

1. **Interconnect saturation ($N \geq 8$):** The central switch becomes a contention point for multicast traffic. With 8 nodes, the 640 Gbps switch bandwidth is fully utilized, causing packet queuing and increased latency.

2. **Memory bandwidth wall ($N \geq 12$):** Multiple FPGAs sharing the same PCIe root complex contend for DDR bandwidth during initial data loading and result retrieval. Our measurements show that with 12 nodes, per-node effective memory bandwidth drops by 31% due to bank conflicts.

3. **Load imbalance ($N \geq 16$):** The workload-balancing scheme (Algorithm S1) becomes less effective as the number of nodes increases, because the smallest indivisible workload (a single CONV layer) becomes too coarse-grained. Fine-grained layer splitting would be required beyond 16 nodes.

For $N \geq 8$, we recommend (i) using multiple switches in a fat-tree topology to alleviate interconnect contention, and (ii) adopting hybrid MPI+PCIe communication to bypass the PCIe switch for inter-FPGA data.

Section S9 Clinical Deployment – Physical Deployment Constraints

For clinical adoption, the physical footprint, power consumption, and cooling requirements must be compatible with standard hospital infrastructure. Our multi-FPGA system is assembled using six XCVU9P

FPGA boards installed in a standard 4U server chassis, accompanied by a 1U network switch—a form factor commonly found in hospital server rooms and imaging suites. Each FPGA board consumes approximately 48 W under full load, with passive heatsinks relying on standard server airflow for cooling, eliminating the need for liquid cooling or specialized ventilation. The total system power draw is under 500 W, well within the capacity of typical 15 A hospital-grade electrical circuits. Operating noise is minimal (< 40 dB) due to the absence of high-speed fans on the FPGA boards, making the system suitable for deployment in or near clinical reading rooms without acoustic disruption. These specifications confirm that our system imposes no special infrastructure requirements and can be readily integrated into existing hospital environments.